U.S.N.A. — Trident Scholar project report; no. 339 (2005)

PARALLEL INTEGER FACTORIZATION USING QUADRATIC FORMS

by

Midshipman Stephen S. McMath, Class of 2005
United States Naval Academy
Annapolis, Maryland

Certification of Advisers Approval

Professor W. David Joyner
Mathematics Department

(signature)

(date)

Assistant Professor Frederick L. Crabbe, IV
Computer Science Department

(signature)

(date)

Acceptance for the Trident Scholar Committee

Professor Joyce E. Shade
Deputy Director of Research & Scholarship

(signature)

(date)

USNA-1531-2

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including g the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
9 May 2005

3. REPORT TYPE AND DATE COVERED

4. TITLE AND SUBTITLE
Parallel integer factorization using quadratic forms

5. FUNDING NUMBERS

6. AUTHOR(S)
McMath, Stephen S. (Stephen Samuel), 1983-

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
US Naval Academy
Annapolis, MD 21402

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
Trident Scholar project report no. 339 (2005)

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
This document has been approved for public release; its distribution is UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT: Factorization is important for both practical and theoretical reasons. In secure digital communication, security of the commonly used RSA public key cryptosystem depends on the difficulty of factoring large integers. In number theory, factoring is of fundamental importance. This research has analyzed algorithms for integer factorization based on continued fractions and binary quadratic forms, focusing on runtime analysis and comparison of parallel implementations of the algorithm. In the process it proved several valuable results about continued fractions. In 1975, Daniel Shanks used class group infrastructure to modify the Morrison-Brillhart algorithm and develop Square Forms Factorization, but he never published his work on this algorithm or provided a proof that it works. This research began by analyzing Square Forms Factorization, formalizing and proving the premises on which the algorithm is based. First, this research analyzed the connections between continued fractions and quadratic forms, proving, among other things, that the square of any ambiguous cycle is the principal cycle. Then, the connection with ideals was developed, requiring a generalization to the standard description and formulas for multiplication of ideals. Lastly, the connection was made with lattices and minima, allowing for a generalization of the formulas relating composition with distance. These results are fundamental to explaining why Square Forms Factorization works. This research also analyzed several variations, including two different parallel implementations, one of which was considered by Shanks and one of which is original. The results suggest that the new implementation, which utilizes composition of quadratic forms, is slower for small numbers of processors, but is more efficient asymptotically as the number of processors grows. Shanks' Square Forms Factorization, including a concept he called Fast Return, has been implemented in C and Magma and some experimental runtime analysis has been done. A parallel version in C has been implemented and tested extensively.
.

14. SUBJECT TERMS: Factorization, Quadratic Form, Daniel Shanks, Continued Fractions, Lattice, Ideal

15. NUMBER OF PAGES
131

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT

18. SECURITY CLASSIFICATION OF THIS PAGE

19. SECURITY CLASSIFICATION OF ABSTRACT

20. LIMITATION OF ABSTRACT

Abstract

Factorization is important for both practical and theoretical reasons. In secure digital communication, security of the commonly used RSA public key cryptosystem depends on the difficulty of factoring large integers. In number theory, factoring is of fundamental importance.

This research has analyzed algorithms for integer factorization based on continued fractions and binary quadratic forms, focusing on runtime analysis and comparison of parallel implementations of the algorithm. In the process it proved several valuable results about continued fractions.

In 1975, Daniel Shanks used class group infrastructure to modify the Morrison-Brillhart algorithm and develop Square Forms Factorization, but he never published his work on this algorithm or provided a proof that it works. This research began by analyzing Square Forms Factorization, formalizing and proving the premises on which the algorithm is based. First, this research analyzed the connections between continued fractions and quadratic forms, proving, among other things, that the square of any ambiguous cycle is the principal cycle. Then, the connection with ideals was developed, requiring a generalization to the standard description and formulas for multiplication of ideals. Lastly, the connection was made with lattices and minima, allowing for a generalization of the formulas relating composition with distance. These results are fundamental to explaining why Square Forms Factorization works.

This research also analyzed several variations, including two different parallel implementations, one of which was considered by Shanks and one of which is original. The results suggest that the new implementation, which utilizes composition of quadratic forms, is slower for small numbers of processors, but is more efficient asymptotically as the number of processors grows.

Shanks' Square Forms Factorization, including a concept he called Fast Return, has been implemented in C and Magma and some experimental runtime analysis has been done. A parallel version in C has been implemented and tested extensively.

Keywords: Factorization, Quadratic Form, Daniel Shanks, Continued Fractions, Lattice, Ideal

Acknowledgements

The thorough analysis of the contributors to this research would be an interesting historical investigation in its own right, but as space is limited, I will merely name a few. This must certainly start with my advisors. Throughout this research, Dr. David Joyner has allowed me to wander the theoretical realms while still keeping me grounded with his insistence on quality proofs of all important results. Dr. Frederick Crabbe, along with providing his computer expertise and understanding of the real world requirements for algorithms, has always found a way of adding a little humor to the mathematical analysis. Thanks must also go to Dr. Duncan Buell and Dr. Hugh Williams for providing me with the unpublished work by Daniel Shanks, along with providing their own research on the subject and aiding in the analysis of some of my work. The recent runtime analysis of Dr. Wagstaff and Jason Gower is greatly appreciated. Thanks must go to Dr. Carl Pomerance for his vision for the parallelization of these algorithms.

My family, although currently distant, has aided this project significantly. I must thank my parents for the curiosity they taught me and my brother LT Daniel McMath for starting me on this problem before I even knew the tools to begin to address it properly.

Lastly, I must thank my God. It is my faith in God that has given me confidence that there is order worth searching for and has given me the hope to keep me going when nothing else could have.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers ... the dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated.
>
> C. F. Gauss [8]

Factorization is important for both practical and theoretical reasons. In secure digital communication, the RSA public key cryptosystem is often used. The security of this cryptosystem depends on the difficulty of factoring large integers. In number theory, factoring is of fundamental importance.

The algorithms for factoring larger and larger integers quickly have developed significantly over the years. There are many ways of doing this, ranging from trial division to the number field sieve. This research focuses on a specific underdeveloped algorithm introduced by Daniel Shank 1975, Square Forms Factorization, but also analyzes two new variations, one due to a conjecture by Pomerance and one introduced by this author. The overall goal of this research was to analyze algorithms for integer factorization based on the use of continued fractions and quadratic forms, focusing on proving mathematical results in these areas and determining average runtime. We proposed several preliminary sub-goals to this end:

1. Analyze the conditions for which Square Forms Factorization provides a factorization.
2. Analyze the connection between continued fractions and quadratic forms and provide related proofs.
3. Analyze a test of direction.
4. Produce a computer implementation of these algorithms.

The first sub-goal was to analyze the conditions for which Square Forms Factorization provides a factorization. This information is important to cryptology, as the security of some public key cryptosystems is dependent on the difficulty of factorization. The conditions for which the algorithms must provide a factorization have been determined but a sufficient set of conditions for which the algorithms will not work has not yet been determined. However, the runtimes vary greatly, so in addition to the original goal, this research analyzed which numbers Square Forms Factorization may factor significantly faster or slower than others, which is just as important since numbers that factor extremely slowly are almost as secure as numbers that the algorithm doesn't factor at all. It is possible to determine sufficient conditions for which a number factors quickly but necessary conditions remain elusive. Specifically, if N is of the form $(a^2m + b)(c^2m + d)$, it is possible to define conditions for special cases, but these conditions have not been generalized. Proposition 1 of §4.3 provides a specific example of this.

The second sub-goal was to analyze the connection between continued fractions and quadratic forms, specifically intending to analyze the work done by Shanks. This research has taken significantly longer than planned for in the original timeline. Much of the existing information on these theories was scattered and disorganized, and many of the papers either lacked proofs or contained errors, so it was valuable to organize this information into a form that could be useful. The most important achievement in this was the development and proof of a formula relating infrastructure distance with composition of quadratic forms.

The third sub-goal was to analyze a test of direction useful to a new algorithm being developed. The research in this area has demonstrated that a test of direction within continued fractions probably cannot be used effectively for factorization.

The fourth of the original sub-goals was to produce a computer implementation of these algorithms. Shanks' Square Forms Factorization has been implemented in both C and Magma, including libraries of functions concerning the operations essential to quadratic forms and continued fractions. This source code will be made available in a distributable form.

Concerning runtime, Jason Gower [9] has recently analyzed the runtime of SQUFOF for his Ph.D. thesis at Purdue University. Included in this was an analysis of multipliers, a concept that will be used in these implementations. However, it has been conjectured by Pomerance that as SQUFOF is easy to parallelize, a parallel implementation may be competitive, so a new goal was set of analyzing the parallel implementation of SQUFOF.

A parallel version of SQUFOF has been developed and has been implemented and tested in C. This code is included in Appendix B. This algorithm has been compared with an older parallel implementation of SQUFOF.

Chapter 2 provides some background to the problem of factorization, basic number theory, and some of the tools related to Square Forms Factorization. Chapter 3 describes how Square Forms Factorization developed from the existing theory. Chapter 4 describes Square Forms Factorization, including several variations of the algorithm, and describes some significance the algorithm may have for cryptology.

Chapter 2

Background

2.1 The Problem and its Importance

There are several different kinds of factorization; this research will focus on integer factorization. Consider an integer N. Factorization is the process of finding integers p and q greater than 1 such that $N = pq$. Complete factorization would require repeating this process for both p and q until all of the remaining factors are prime. However, if an algorithm can be developed to quickly factor N into p and q, the same algorithm can be used over again on p and q. For example, it is easy to see that $105 = 5 \cdot 21$ and then repeat to factor 21. From here, you would see that since $21 = 3 \cdot 7$, $105 = 5 \cdot 3 \cdot 7$. Although in this simple case, the complete factorization is easy to find, this task becomes much harder for large numbers.

2.1.1 Number Theoretic Applications

In number theory, the factors of a number dictate many of the characteristics of the number. For example, Euler's ϕ function, which tells how many numbers less than N are relatively prime to N, can be directly calculated from the complete factorization. For an integer that is the product of distinct odd primes, $\phi(N)$ may be calculated by multiplying each of the factors minus 1.

Example 1. $105 = 5 \cdot 3 \cdot 7$. Therefore $\phi(105) = (5 - 1)(3 - 1)(7 - 1) = 48$, so there are 48 integers less than 105 that are relatively prime to 105.

Also, determining whether a number is a *quadratic residue* (i.e. the square of another number modulo N), can be determined directly using Gauss's Quadratic Reciprocity Law if the complete factorization of N is known.

Example 2. $19^2 = 361 = 46 + 3 \cdot 105$, so that 46 is a quadratic residue modulo 105 with square root 19. One would represent this as $19^2 \equiv 46 \pmod{105}$. With the complete factorization, we can use Gauss's Quadratic Reciprocity Law to analyze 46 modulo 3,5, and 7 to determine whether or not 46 is a quadratic residue modulo 105 without actually having to find its square root first [8].

Therefore, factoring large numbers has been a focus of research for a variety of theoretical reasons.

2.1.2 Cryptographic Applications

One practical application of factorization is public key *cryptography.* The idea of public key cryptography is that it is possible to keep a communication secret without having to keep the key secret. The message is encrypted by one key, which is made public, and is decrypted by another key, which is kept secret. In the most prevalent public key encryption system, *RSA*[1] [19], the cryptographer chooses a number that is the product of two large prime[2] numbers p and q: $N = pq$. Then an exponent e is chosen such that e is relatively prime to $(p-1)(q-1)$. Although there are several variations, in the normal public key version, N and e are made public. The user of the RSA system then privately calculates d, the inverse of e modulo $(p-1)(q-1)$. Anyone is able to encrypt something to him by raising blocks of the message to the power e, modulo N: $c \equiv m^e \pmod{N}$, where m is the original message and c is the encrypted message. Then, the recipient is able to decrypt by evaluating $m \equiv c^d \pmod{N}$ [19].

[1] RSA is named after Rivest, Shamir, and Adleman. It was earlier developed by Clifford Cooks of GCHQ, but this was only recently declassified [5].

[2] Usually, these are just numbers that pass several primality tests and thus have a high probability of being prime, called pseudo-primes. Very rarely, one of them will not be, but this is rare enough to not cause significant problems.

Example 3.

$$p = 4327, \ q = 1009, \ N = pq = 4365943$$

$$e = 2005$$

and from this the cryptographer would use p, q and e to calculate $d = 865597$. He would make N and e public. Suppose someone wants to send him the message 809.

The sender would evaluate $809^{2005} \pmod{N}$ as 1591327. This would be sent over the internet.

The receiving computer would then evaluate $1591327^{865597} \pmod{N}$ as 809. Since this is the only computer with access to d, anyone else intercepting the message would have great difficulty in determining the message

Another application of public key cryptography is for signatures. If the sender uses his private key to encrypt a message, the receiver may use the public key to decrypt the message and be certain of who the message originated from and that no one along the way has modified it. Often this is combined with a second layer of encryption to prevent anyone else from also being able to use the public key to read the message.

As long as someone intercepting a message is unable to factor N, it is usually impossible to obtain d, so that the message cannot be broken. The security of RSA and its variations depends highly on whether or not N can be factored [24]. Although extremely fast factorization would be a threat to these systems, the advances in number theory produced by faster factorization would likely provide a number of alternative secure systems. Also, in addition to the potential for alternative secure systems, there is the strong possibility that fast factorization algorithms will work better for some numbers than others. If there are classes of numbers that a faster factorization algorithm does not work on, this would enable designers of the algorithm to increase their security by relying more on these numbers. Regardless of whether or not the algorithm works for all numbers or provides alternative systems, for security purposes it is necessary to understand the strengths and weaknesses of the algorithm.

2.1.3 Runtime Analysis

Up to this point we have referred to the speed of factorization in general terms, but there are several different ways to classify the speed of an algorithm. Let N be the number to factor. Let n be the number of bits in N, $n = \log_2 N$. An algorithm's run time is called *exponential* if it increases exponentially with the size of the input, in this case n. *Linear* refers to an algorithm where the time increases proportionally to the number of bits[3]. *Polynomial* refers to an algorithm for which the time required is some polynomial function of n. Thus, linear time is a special case of polynomial time. There are some algorithms that fall in between polynomial and exponential time and are referred to as sub-exponential.

Note that different algorithms may be faster for different sizes of numbers or even for different systems. This is why the runtime is analyzed in terms of growth. For small values of n, linear and exponential may be fairly close or linear may even be faster, but the runtime of an exponential algorithm will grow faster with the size of n, so that for sufficiently large n, an exponential algorithm will be slower than a sub-exponential, which will be faster than a polynomial, which will be faster than a linear.

The runtimes for sub-exponential algorithms are described by complicated formulas. Define

$$L(\alpha, c) = \exp(cn^{\alpha}(\log n)^{1-\alpha}))$$

If $\alpha = 1$, this is an exponential algorithm. If $\alpha = 0$, this is a polynomial algorithm. For values of α in between 0 and 1, the algorithms are sub-exponential.

In terms of asymptotic average runtime, the best general purpose factorization algorithm is the *general number field sieve*, with a runtime[4] of $L(1/3, \frac{4}{3^{2/3}} + o(1))$ [6]. Two other algorithms are currently in common use, the *elliptic curve method*, which has a runtime of $L(1/2, 1 + o(1))$ [13] and the *multiple polynomial quadratic sieve*, which has a runtime of

[3] Since $n = \log_2 N$, so that such a runtime is logarithmic in N, this is often referred to as logarithmic, resulting in a certain amount of confusion. Note that both ways of expressing runtime will be used in this paper.

[4] The expression $o(1)$ is *little o* notation. It converges to 0 as n goes to infinity. An understanding of little o notation is not important for understanding this report.

$L(1/2, 1)$ [16]. Although each of these algorithms is slower on average, they are each faster for some types of integers, so that in combination they tend to be faster than the number field sieve.

Another less commonly considered characteristic of a factorization algorithm is the efficiency with which it can use multiple processors working simultaneously. The current trend in ultra-fast computing is to use a large number of inexpensive processors instead of a single expensive processor. Given this trend, it is important that a good algorithm be able to use a multiple processor system efficiently. Dividing a single task between multiple processors is called *parallelization*. With perfect efficiency, factoring a number with 10 processors should take on average one tenth the time required for factorization on a single processor, but no algorithm can ever quite attain perfect efficiency.

2.2 Basic Number Theory

> Mathematics is the Queen of the sciences, and arithmetic the Queen of mathematics.
>
> C. F. Gauss [25]

This section provides a quick overview of basic number theory concepts and notation that will be used throughout this report. Those already familiar with number theory are welcome to skip this section, as none of this is original and may be found in any standard number theory text [10].

Number theory analyzes sets of numbers. If an element e is in a set S, this is written $e \in S$. If an element e is not in a set S, this is written $e \notin S$. If all the elements in set S_1 are in set S_2, then S_1 is contained in set S_2 and this is written $S_1 \subset S_2$. If $S_1 \subset S_2$ and $S_2 \subset S_1$, then $S_1 = S_2$.

The set of all real numbers is denoted $\mathbb{R}$. Although this set will be important, it is primarily several special subsets of the real numbers that this research will focus on:

The first subset of the real numbers is the set of integers, represented as $\mathbb{Z}$.

$$\mathbb{Z} = \{... - 3, -2, -1, 0, 1, 2, 3...\}.$$

The natural numbers are the integers ≥ 1. The operations on the integers are the standard addition and multiplication. Addition has the usual four basic properties: associativity, commutativity, identity, and inverses.

Multiplication has three basic properties: associativity, commutativity, and identity.

One other property, distributivity, involves both addition and multiplication:

$a \cdot (b + c) = ab + ac.$

There are several other relations that are important in $\mathbb{Z}$. A natural number a divides an integer b if there exists an integer k such that $b = ka$ and this is written $a \mid b$. If a does not divide b, this is written $a \nmid b$. A natural number > 1 is *prime* if it may only be divided by itself and 1. There are several basic properties of this relation:

If $a \mid b$ and $b \mid a$, then $a = b$.

If $a \mid b$ and $a \mid c$, then $a \mid (b + c)$.

If $a \mid b$, then $a \mid bc$.

If a is prime and $a \mid bc$, then $a \mid b$ or $a \mid c$.

If for some integer n and a prime p, $p^n \mid b$, but $p^{n+1} \nmid b$, this is written $p^n \parallel b$.

c is the *greatest common divisor* of a and b if $c > 0$, $c \mid a$, $c \mid b$, and for any integer d such that $d \mid a$ and $d \mid b$, $d \mid c$. This is written $c = \gcd(a, b)$ or sometimes (although not in this report) as merely (a, b).

Two integers a and b are *relatively prime* if $\gcd(a, b) = 1$, that is, if a and b have no factor in common.

If p, m, n are integers with p prime such that $p^m \parallel a$ and $p^n \parallel b$, then $p^{\min(m,n)} \parallel \gcd(a, b)$.

If $c = \gcd(a, b)$, there exist integers $x, y \in \mathbb{Z}$ such that $ax + by = c$. $\gcd(a, b)$, along with these two integers, may be efficiently calculated using Euclid's extended algorithm.

c is the *least common multiple* of a and b if $c > 0$, $a \mid c$, $b \mid c$, and for any integer d such that $a \mid d$ and $b \mid d$, $c \mid d$. This is written $c = \operatorname{lcm}(a, b)$ or sometimes merely as $\{a, b\}$. If p, m, n are integers with p prime such that $p^m \parallel a$ and $p^n \parallel b$, then $p^{\max(m,n)} \parallel \operatorname{lcm}(a, b)$.

The gcd and lcm are related by the formula $\gcd(a, b) \cdot \operatorname{lcm}(a, b) = ab$.

If $c \mid (a - b)$, then we say that a is *congruent* to b modulo c, written $a \equiv b \pmod{c}$. This relation has several basic properties:

If $a \equiv b \pmod{n}$, then $b \equiv a \pmod{n}$.

If $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$, then $a \equiv c \pmod{n}$.

If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $a + b \equiv c + d \pmod{n}$ and $ac \equiv bd \pmod{n}$.

By using Euclid's extended algorithm to compute greatest common divisors, for a relatively prime to b it is possible to calculate $a^{-1} \pmod{b}$. Thus, division modulo b is defined as reducing to least terms and then multiplying by the inverse.

One important type of number is a *quadratic residue.* The integer a is a quadratic residue of integer base b if there exists some integer x such that $x^2 \equiv a \pmod{b}$. For example, 2 is a quadratic residue of 7 because $3^2 = 9 \equiv 2 \pmod{7}$. The symbol used is $\left(\frac{a}{b}\right)$. $\left(\frac{a}{b}\right) = 1$ if $x^2 \equiv a \pmod{b}$ has a solution, -1 if it does not, and 0 if a and b are not relatively prime. If a is a quadratic residue of b and a is a quadratic residue of c and b and c are relatively prime, then a is a quadratic residue of bc. Gauss's Quadratic Reciprocity Law provides a fast way of determining quadratic residues modulo a base whose prime factorization is known:

Theorem 2.2.1. *For p and q distinct primes:*

$$\left(\frac{p}{q}\right)(\frac{q}{p}) = (-1)^{(p-1)(q-1)/4}$$

$$\left(\frac{2}{p}\right) = (-1)^{(p^2-1)/8}$$

$$\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2}$$

There are many other well-known properties concerning congruences. For example, "Fermat's little theorem": for p prime and a an integer if $p \nmid a$, then $a^{p-1} \equiv 1 \pmod{p}$. This is often used to test whether not an integer N is prime. (If for some $a < N$, $a^{N-1} \not\equiv 1$

(mod N), then N is not prime. However, there are infinitely many composite numbers that will appear prime for any a such that $\gcd(a, N) = 1$, so the converse is not necessarily true [1].)

In $\mathbb{Z}$, inverses are not defined for multiplication. However, this question brings us to the set of rational numbers, represented as $\mathbb{Q}$.

$$\mathbb{Q} = \{a/b : a, b \in \mathbb{Z}, b \neq 0\}$$

All of the properties for addition and multiplication in $\mathbb{Z}$ still apply, in addition to one more:

If $a \in \mathbb{Q}$ and $a \neq 0$, then there exists a unique integer b such that $a \cdot b = 1$. This is written as a^{-1} or $1/a$.

Let α denote a real or complex number not belonging to $\mathbb{Q}$. Denote the set of all finite linear combinations with integer coefficients of all the non-negative powers of α by,

$$\mathbb{Z}[\alpha] = \{a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3 + ... : a_i \in \mathbb{Z}\}.$$

For example, if $\alpha = \sqrt{2}$, then $\mathbb{Z}[\sqrt{2}]$ contains $5 + 3\sqrt{2}$ and $4 - 2\sqrt{2}$, but does not contain $\frac{3}{\sqrt{2}}$.

Alternately, the new set could be the set of all linear combinations of all the powers of α. That is,

$$\mathbb{Q}(\alpha) = \{...a_{-2}\alpha^{-2} + a_{-1}\alpha^{-1} + a_0 + a_1\alpha + a_2\alpha^2 + ... : a_i \in \mathbb{Q}\}$$

For example, $\mathbb{Q}(\sqrt{2})$ contains both $3 + \sqrt{2}$ and $4 + \frac{2}{\sqrt{2}}$, but does not contain $\sqrt{3}$.

For an element ζ in either of the sets $\mathbb{Z}[\sqrt{N}]$ and $\mathbb{Q}(\sqrt{N})$, $\overline{\zeta}$ refers to the *conjugate* of ζ, which is found by changing the sign of the algebraic part. For example, if $\zeta = 1 + \sqrt{3}$, then $\overline{\zeta} = 1 - \sqrt{3}$.

The *norm* of ζ is $\mathcal{N}(\zeta) = \zeta\overline{\zeta}$.

2.3 Quadratic Field Number Theory

This section introduces four areas of mathematics that this research has focused on: continued fractions, quadratic forms, ideals, and lattices. These areas are closely related and therefore have the same property of being periodic. See the appendices for a more detailed analysis of each of these topics, along with related proofs and an analysis of their connections.

1. A *continued fraction* is a tool originally used for rational approximation. Given a number α, the goal is to represent α in the form

   $$\alpha = b_0 + \frac{1}{b_1 + \frac{1}{b_2 + ...}}$$

   for integers b_i. This is often abbreviated as $[b_0, b_1, b_2, ...]$. The expressions found by truncating this, $b_0, [b_0, b_1], [b_0, b_1, b_2], ...$ are called the *convergents.* These simplify down to rational numbers A_n/B_n.

   The sequence of b_i's is found by the recursive formulas

   $$x_0 = \alpha,\ b_0 = \lfloor x_0 \rfloor \tag{2.1}$$

   $$\forall i \geq 1\ x_i = \frac{1}{x_{i-1} - b_{i-1}},\ b_i = \lfloor x_i \rfloor \tag{2.2}$$

   Note that this formula is derived by solving the equation $x_{i-1} = b_{i-1} + \frac{1}{x_i}$ for x_i. $\lfloor x \rfloor$ refers to the *floor* of x, the greatest integer less than or equal to x.

   If $x_0 = \sqrt{N}$ for N a non-square integer (that is, not the square of another integer), then this sequence may be calculated efficiently. In particular, each x_i reduces to the form $\frac{\sqrt{N}+P}{Q}$ with P and Q integers. The integer P in the numerator is called the *residue.* The denominator Q is called a *pseudo-square.* There are formulas to calculate these recursively, after the first several steps, without doing arithmetic on any integer larger than $\sqrt{N}$:

$$b_i = \left\lfloor \frac{\lfloor\sqrt{N}\rfloor + P_{i-1}}{Q_i} \right\rfloor \quad P_i = b_i Q_i - P_{i-1} \quad Q_{i+1} = Q_{i-1} + b_i(P_{i-1} - P_i) \tag{2.3}$$

The rational approximation produced may also be evaluated recursively, so that the integers A_i and B_i such that $A_i/B_i = [b_0, b_1, ...b_i]$ may be calculated efficiently. For factorization, the relation

$$A_{i-1}^2 \equiv (-1)^i Q_i \pmod{N} \tag{2.4}$$

has been important for several algorithms. This equation also explains the name ‘pseudo-square’.

Take the following example:

Example 4.

$$\begin{array}{rr}
x_0 = \sqrt{41},\ b_0 = 6 & \sqrt{41} = 6 + \frac{1}{x_1} \\
x_1 = \frac{1}{x_0 - b_0} = \frac{1}{\sqrt{41}-6} = \frac{\sqrt{41}+6}{5},\ b_1 = 2 & \sqrt{41} = 6 + \frac{1}{2+\frac{1}{x_2}} \\
x_2 = \frac{1}{x_1 - b_1} = \frac{5}{\sqrt{41}-4} = \frac{\sqrt{41}+4}{5},\ b_2 = 2 & \sqrt{41} = 6 + \frac{1}{2+\frac{1}{2+\frac{1}{x_3}}} \\
x_3 = \frac{1}{x_2 - b_2} = \frac{5}{\sqrt{41}-6} = \frac{\sqrt{41}+6}{1},\ b_3 = 12 & \sqrt{41} = 6 + \frac{1}{2+\frac{1}{2+\frac{1}{12+\frac{1}{x_4}}}}
\end{array}$$

At this point it is evident that $x_4 = x_1$ and the sequence is periodic. This always occurs with the continued fraction expansion for $\sqrt{N}$ for N non-square. The length of this period is closely related to several deep number theory ideas involving the class number and the regulator.

A more thorough explanation of continued fractions, including detailed proofs is given in Appendix A.1.

2. A *binary quadratic form* is a polynomial of the form $F(x, y) = ax^2 + bxy + cy^2$, for $a, b, c \in \mathbb{Z}$. (Often this is abbreviated[5] as (a, b, c)).

[5] As b is usually even, many writings on quadratic forms represent this as the triplet $(a, b/2, c)$, so that

Two quadratic forms are *equivalent* if they have the same range, the set of possible values for $F(x, y)$ for integer values of x and y, written $F_1 \sim F_2$. Appendix A.2 gives a more precise description of equivalence. The surprising fact is that two quadratic forms are equivalent if and only if they correspond to terms from the same continued fraction expansion (Theorem A.2.4 in Appendix A.2), so that the cycle formed by the x_i's used to compute continued fractions corresponds to a cycle of equivalent quadratic forms.

The primary operation on quadratic forms is *composition.* This operation is defined in detail in Appendix A.2, Proposition 2. The composition of two binary quadratic forms is another binary quadratic form, written with the symbol $(*)$. Note that although the original definition of this operation is related to multiplication, this is an operation that is quite distinct from multiplication, as the product of two binary quadratic forms is no longer a binary quadratic form.

Equivalence and composition are closely related. Specifically, if $F_1 \sim F_2$, then $F_1 * G \sim F_2 * G$. The cycles of equivalent forms are the elements of the *class group*, which has been studied extensively.

A more thorough explanation of quadratic forms is given in Appendix A.2.

3. A *lattice* is the set of all finite linear combinations (with integer coefficients) of a generating set that contains a vector space basis for the rational vector space the lattice is contained in. For this research, the ambient vector space is $\mathbb{Q}(\sqrt{N}) \times \mathbb{Q}(\sqrt{N})$. If $\alpha_1, \alpha_2, ..., \alpha_k \in \mathbb{Q}(\sqrt{N}) \times \mathbb{Q}(\sqrt{N})$ are vectors, then

$$[\alpha_1, \alpha_2, ..., \alpha_k] = \left\{ \sum_{i=1}^{k} n_i \alpha_i : n_i \in \mathbb{Z} \right\}$$

is a lattice.

The notation is often abused slightly, so that the vector $\langle \zeta, \overline{\zeta} \rangle$ is represented by just ζ.

$-14x^2 + 10xy + 5y^2$ would be represented $(-14, 5, 5)$. This report does not take out this factor of 2. Although this will be clear throughout this report, please observe the potential for confusion on this "annoying little 2 [2]" when comparing different sources on quadratic forms.

There are several properties of lattices that are important for this research.

Definition 2.3.1. For a vector $v = \langle v_1, v_2 \rangle$, the *normed body* of v, $\mathcal{R}(v)$ is the set

$$\mathcal{R}(v) = \{\langle x_1, x_2 \rangle : x_1, x_2 \in \mathbb{R}, |x_1| < |v_1|, |x_2| < |v_2|\}$$

Abusing notation again, denote $\mathcal{R}(\xi) = \mathcal{R}(\langle \xi, \bar{\xi} \rangle)$.

A number ξ (or actually the corresponding vector) is a *minimum* of $\mathcal{L}$ if $\mathcal{R}(\xi) \cap \mathcal{L} = \{0\}$, where 0 is the vector $\langle 0, 0 \rangle$. Figure 2.1 provides an example of this.

A lattice $\mathcal{L}$ is *reduced* if $1 \in \mathcal{L}$ and 1 is a minimum.

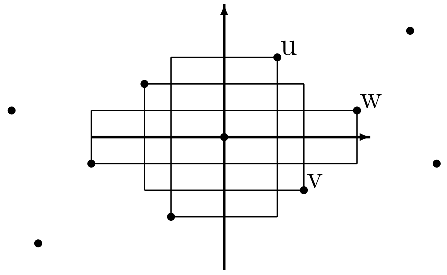

Figure 2.1: A Lattice with Minima

The image above demonstrates minima of a lattice. The lattice is the set of points. Each of the points u, v, and w are minima of this lattice, since there are no points of the lattice inside their respective normed bodies (the rectangles) except the origin.

Another important characteristic of minima is adjacency. If $\langle x_1, y_1 \rangle$ and $\langle x_2, y_2 \rangle$ are minima with $|x_1| > |x_2|$ and $|y_1| < |y_2|$, these two minima are *adjacent* if there does not exist another minima $\langle x_3, y_3 \rangle$ such that $|x_2| < |x_3| < |x_1|$ and $|y_1| < |y_3| < |y_2|$. In Figure 2.1, u and v are adjacent minima and v and w are adjacent minima.

The important connection between lattices and continued fractions is a method developed by Voronoi for finding a chain of adjacent minima for a lattice (and a chain of other reduced lattices). This process matches the continued fraction algorithm exactly, a property that implies that the distance derived from lattices also applies to continued fractions and quadratic forms.

A more thorough description of lattices is given in Appendix A.4.

4. An ideal is a common algebra concept that is used in a variety of contexts. For a ring S , a set I is an *ideal* of S if

   $I \subset S$.

   If $a, b \in I$, then $a + b \in I$ and $a - b \in I$.

   If $a \in I$ and $s \in S$, then $a \cdot s \in I$.

   A classic example is the set of all even integers, $2\mathbb{Z}$, within the set $\mathbb{Z}$ of all integers. The sum of two even integers is still an even integer and the product of an even integer with any integer is an even integer.

   This analysis focuses on ideals of $\mathbb{Z}[\sqrt{N}]$, for N a non-square positive integer. Describing these ideals will again require the notation for the lattice generated by a set. With this notation the ideals of $\mathbb{Z}[\sqrt{N}]$ are of the form $[Q, s\sqrt{N}+P]$, for some integers Q, s, P and a set $I = [Q, s\sqrt{N} + P]$ is an ideal of $\mathbb{Z}[\sqrt{N}]$ if and only if $sQ \mid \mathcal{N}(s\sqrt{N} + P)$, $s \mid Q$, and $s \mid P$ (Appendix A.3, Lemma A.3.2). An ideal is *primitive* if $s = 1$.

   The multiplication of ideals is important to many fields. For the ideals $I = [Q, \sqrt{N}+P]$ and $J = [Q', \sqrt{N} + P']$,

   $$I \cdot J = [QQ', Q\sqrt{N} + QP', Q'\sqrt{N} + Q'P, (\sqrt{N} + P)(\sqrt{N} + P)]$$

   This may then be simplified to reduce it to an ideal of the form $s[q, \sqrt{N} + p]$, where s, q, and p may be calculated efficiently.

The quadratic form (of discriminant $\Delta \equiv 0 \pmod 4$) $F(x,y) = Ax^2 + Bxy + Cy^2$ corresponds to the ideal

$$\left[A, \sqrt{\left(\frac{B}{2}\right)^2 - AC} + \frac{B}{2}\right]$$

and similarly the ideal $[Q, \sqrt{N} + P]$ corresponds to the quadratic form

$$F(x,y) = Qx^2 + 2Pxy + \left(\frac{P^2 - N}{Q}\right) y^2$$

Note that $\Delta = 4N$.

The value of ideals is that the equations defining multiplication of ideals correspond exactly to the equation defining composition of quadratic forms, providing a connection between quadratic forms and lattices. A more thorough description of ideals is given in Appendix A.3.

The appendices also describe the one to one correspondence between the elements of these different sets. For this paper, it is sufficient to understand that these maps do exist, that is, each term in the continued fraction expansion is paired with exactly one quadratic form, exactly one lattice, and exactly one ideal and given any one, any of these four expressions may be calculated immediately. The indices derived from continued fractions are typically also used for the related quadratic forms, ideals, and lattices.

As all of these tools are closely related, they all have the property of being periodic, as demonstrated by Example 4. The reason for this is that the bounds on the coefficients of a reduced quadratic form (Definition A.2.1) imply that there can only be a finite number of them. Since the recursive formulas (2.3) provide a way of getting from one reduced quadratic form to the next, this process must eventually repeat and thus be periodic. The length of this period is equal to the regulator of the class group and is related to many other number theory concepts ([12], [20]).

If the continued fraction expansion is started with something other than $x_0 = \sqrt{N}$, the result may be a sequence that is completely disjoint from the *principal* cycle (the sequence that begins with $x_0 = \sqrt{N}$). Most of these cycles are symmetric[6]. Lemma A.1.6 and Theorem A.1.7 provide a thorough definition and analysis of these symmetries, but the concept can be readily understood from several examples:

Example 5. In Example 4, the period of the denominators was $1, 5, 5, 1, ...$, so that 1 and 5 are both *symmetry points*. If $x_0 = \sqrt{115}$, then the period of the denominators is

$$1, 15, 6, 11, 9, 10, 9, 11, 6, 15, 1, ...$$

so that the symmetry points are 1 and 10.

If instead $x_0 = \frac{\sqrt{115}+9}{2}$, then the period of the denominators is

$$2, 17, 3, 5, 3, 17, 2, ...$$

This is a different cycle for the class group for $N = 115$.

These are the cycles that are important for factorization.

The square of a quadratic form in the principal cycle is also in the principal cycle and Theorem A.2.8 shows that the square of a form in any ambiguous cycle is also in the principle cycle. This property enables a variety of methods of finding points in ambiguous cycles other than the principal cycle.

[6] The non-symmetric cycles are not analyzed in this paper.

Chapter 3

History: From Morrison and Brillhart to Present

3.1 Before Shanks

In 1931, D. H. Lehmer and R. E. Powers [11] introduced one simple and fairly intuitive algorithm for using continued fractions for factorization. Unfortunately, as it had the frustrating tendency to fail a few times before succeeding, it was dropped. However, as better computing capabilities became available, John Brillhart began to consider that although this algorithm was cumbersome for smaller numbers, it might have an advantage for larger numbers. In 1970, Morrison and Brillhart implemented CFRAC, as it became known, and tried to factor a 39 digit number [15]. The entire algorithm is a bit more complicated, but here is a description sufficient for our purposes.

If the equation

$$x^2 \equiv y^2 \pmod{N} \tag{3.1}$$

can be solved such that

$$x \not\equiv \pm y \pmod{N}, \tag{3.2}$$

then $\gcd(x-y, N)$ provides a nontrivial factor of N. Choosing a value for x and then looking for a value for y that satisfies equations (3.1) and (3.2) is not computationally effective for large N. Fermat, the first to employ this concept, tested values of x greater than $\sqrt{N}$ to find a value such that $x^2 \pmod N$ was already a perfect square. However, these numbers get large quickly. Continued fractions provide another approach to achieving this and since $0 < Q_i < 2\sqrt{N}$, the chances of finding a perfect square are greatly improved. From (2.4), $A_{i-1}^2 \equiv (-1)^i Q_i \pmod N$. If for some i, Q_{2i} is a perfect square then this provides a solution to (3.1) and it only remains to check whether or not $\gcd(x+y, N)$ or $\gcd(x-y, N)$ provide a nontrivial factor of N. However, there are not very many perfect squares in the continued fraction expansion so Morrison and Brillhart [15] used products to obtain squares. For example, for $N = 1333$, the continued fraction expansion provides $73^2 \equiv -3 \pmod N$ and $1789^2 \equiv -12 \pmod N$. From this, $(73 \cdot 1789)^2 \equiv (-3)(-12) = 6^2 \pmod N$, quickly yielding $\gcd(73 \cdot 1789 - 6, 1333) = 43$, so that $1333 = 31 \cdot 43$.

There are two problems with this algorithm. First, it requires the calculation of the A_i's, which are of the same size as N, after reduction modulo N, while the rest of the algorithm only requires arithmetic on numbers of size $\sqrt{N}$. Second, after going through a nontrivial amount of computation to find a relation that solves (3.1), not all of these result in a factorization. I provide one example:

Example 6. In the continued fraction for $\sqrt{1333}$, $Q_6 = 9 = 3^2$ and $A_5 = 10661$, so that $10661^2 \equiv 3^2 \pmod{1333}$. Unfortunately $10661 \equiv -3 \pmod{1333}$, so this square does not result in a nontrivial factor of 1333.

3.2 Shanks' Work

In 1975, Daniel Shanks developed several very interesting algorithms for factorization ([22],[23]) based on an understanding of quadratic forms and the class group infrastructure. First he developed an improvement to the Morrison-Brillhart algorithm. Roughly speaking, rather then saving the A_i's, he was able to use composition of quadratic forms to combine numbers to produce squares and then use the "infrastructure" to use those squares to find a factorization.

In addition, he developed from the concept of infrastructure a system of predicting whether or not any given square would provide a nontrivial factor. Unfortunately, this didn't save very much time and was a much more complicated algorithm than the Morrison-Brillhart algorithm.

From here the development of the algorithm was prompted by the number $2^{60}+2^{30}-1$. It failed a Fermat primality test[1], but when Morrison and Brillhart tried to factor it, it failed 114 times. Therefore, they stopped, multiplied it by some small constant and tried again. This time it worked on the first try, but they wanted to know why it had failed so many times. So they asked Shanks to analyze it. Unfortunately (or fortunately in hindsight), Shanks only had an HP-65 available and he couldn't fit his entire algorithm into it. Therefore, he discarded all the work of combining numbers to form squares and just cycled through until he found one already there. The code for this was much shorter, and as it turned out the algorithm, which became known as Square Forms Factorization or SQUFOF, was actually significantly faster. Here's how it works:

The continued fraction cycles contain symmetry points that potentially provide a factorization for N (Lemmas A.1.6-A.1.7). The goal of SQUFOF and any related algorithms is to search for these symmetry points. SQUFOF, described in Figure 3.1, provides a fairly simple way of achieving this. The basic version searches the sequence in order until it finds a perfect square on an even index[2]. Then it takes the square root of the denominator and the conjugate of the numerator. According to Theorem A.2.8, this new term still corresponds to an ambiguous cycle, that is, it will still have two symmetry points. By Theorem A.5.2, one of these symmetries point is behind this term (in front of it after taking the conjugate) a distance that is half of the distance from the start to the square found. Therefore, the algorithm continues cycling from here until it finds the symmetry point, which is indicated by a repeated numerator. At this point P_i must then have a factor in common with Q_i because

[1] The Fermat primality test is based on Fermat's classical result that for p prime, $a^p \equiv a \pmod{p}$ [10]. Equivalently, if $a^N \not\equiv a \pmod{N}$ for some integer a, then N isn't prime.

[2] At worst case, the 1 at the end of the cycle will provide this. The analysis by Shanks [23] and later by Gowers [9] predict that for N with $k+1$ distinct factors, there should be on average $\ln(8)\frac{2+\sqrt{2}}{4}\frac{\sqrt[4]{N}}{2^k-1}$ steps until the first square that provides a non-trivial factor.

of the formula $P_i = b_iQ_i - P_{i-1}$ (see Case 1 of Theorem A.1.7). Since $N = P_i^2 + Q_iQ_{i+1}$, this common factor is a factor of $2N$ and may be found by taking the greatest common divisor of P_i and Q_i. For an analysis of how Shanks predicted when the factor found would be non-trivial, see §A.6. With the inclusion of an idea Shanks called Fast Return, explained shortly, these predictions will be unnecessary.

```
Given $N$ to be factored:
$x_0 \leftarrow \sqrt{N}$ $b_0 \leftarrow \lfloor x_0 \rfloor$
while $Q_i \neq$ perfect square
    Apply equation(2.2) twice
    Reduce $x_i$ to the form $\frac{\sqrt{N}-P_i}{Q_i}$
$x_0' \leftarrow \frac{\sqrt{N}+P_i}{\sqrt{Q_i}}$
while $P_i \neq P_{i-1}$
    Apply equation (2.2) and reduce

gcd($Q_i, P_i$), the greatest common
divisor, is a nontrivial factor of N.
```

Figure 3.1: SQUFOF without Composition

Example 7. Let N be 1353:

$$x_0 = \sqrt{1353}\ b_0 = 36$$

$$x_1 = \frac{1}{\sqrt{1353}-36} = \frac{\sqrt{1353}+36}{57} = 1 + \frac{\sqrt{1353}-21}{57}$$

$$x_2 = \frac{57}{\sqrt{1353}-21} = \frac{\sqrt{1353}+21}{16} = 3 + \frac{\sqrt{1353}-27}{16} \tag{3.3}$$

The second fraction in each step is found by rationalizing. At each step, the integers taken out are b_i and the remaining fractions are between 0 and 1. After subtracting b_i the remaining fraction is inverted to find x_{i+1}. As a point of reference, we have approximated so far that $\sqrt{1353} \approx 36 + \frac{1}{1+\frac{1}{3}}$. SQUFOF stops here because 16 is a perfect square. Taking the conjugate of the numerator and the square root of the denominator, we obtain:

$$x_0' = \frac{\sqrt{1353}+27}{4} = 15 + \frac{\sqrt{1353}-33}{4}$$

$$x_1' = \frac{4}{\sqrt{1353}-33} = \frac{\sqrt{1353}+33}{66} = 1 + \frac{\sqrt{1353}-33}{66} \tag{3.4}$$

Here, since the residue 33 is repeated, we quickly find that 33 is a factor of 1353. $1353 = 33 \cdot 41 = 3 \cdot 11 \cdot 41$.

Shanks developed Square Forms Factorization, or SQUFOF, based on a concept called the *infrastructure* of the class group. This refers to the relationship between distance and composition. Define the *distance* from F_m to F_n by:

$$D(F_m, F_n) = \log(\prod_{k=m+1}^{n} x_k) \tag{3.5}$$

where x_k are the corresponding terms in the continued fraction expansion. Since distance is the log of the product of the terms in between, roughly speaking, distance is nearly proportional to the difference in indices. Then the location of the form resulting from the composition of two quadratic forms is controlled by:

$$D(F_1 * G_1, F_n * G_m) = D(F_1, F_n) + D(G_1, G_n) + \zeta \tag{3.6}$$

where ζ is small[3]. This formula is proven in the arguments leading up to Theorem A.5.2.

By equation (3.6), the distance from a symmetry point is doubled when a quadratic form is squared, so that the square root of a quadratic form is half the distance from a symmetry point as the original quadratic form. The change made to the continued fraction expansion

[3] The bound for ζ is proportional to $\log\log N$ while the two distances on the right side of the formula have a bound proportional to $\log N$, so ζ is usually negligible, although it may be calculated if necessary.

upon the occurrence of a perfect square corresponds exactly to taking the square root of a quadratic form and reversing its direction, so that in the second phase (after the change in sequence is made) the algorithm is going backwards on a different cycle. The distance covered from making this change until obtaining the symmetry point will be one half the distance covered before finding this square. Since this distance may be known as well as necessary, it is possibly to use some of the quadratic forms found along the way to shorten this return.

Formally, here is the version of Square Forms Factorization that Shanks intended:

```
Given $N$ to be factored:
$Q_0 \leftarrow 1, P_0 \leftarrow \lfloor\sqrt{N}\rfloor, Q_1 \leftarrow N - P_0^2$
$r \leftarrow \lfloor\sqrt{N}\rfloor$
**while** $Q_i \neq$ perfect square for some $i$ even
    $b_i \leftarrow \left\lfloor \frac{r+P_{i-1}}{Q_i} \right\rfloor$
    $P_i \leftarrow b_i Q_i - P_{i-1}$
    $Q_{i+1} \leftarrow Q_{i-1} + b_i(P_{i-1} - P_i)$
    **if** $i = 2^n$ for some $n$
        Store $(Q_i, 2 \cdot P_i)$ $F_0 = (\sqrt{Q_i}, 2 \cdot P_{i-1}, \frac{P_{i-1}^2 - N}{Q_i})$
Compose $F_0$ with stored forms according to the
binary representation of $i/2$ and store result to $F_0$.
$F_0 = (A, B, C)$
$Q_0 \leftarrow |A|, P_0 \leftarrow B/2, Q_1 \leftarrow |C|$
$q_0 \leftarrow Q_1, p_0 \leftarrow P_0, q_1 \leftarrow Q_0$
**while** $P_i \neq P_{i-1}$ and $p_i \neq p_{i-1}$
    Apply same recursive formulas to $(Q_0, P_0, Q_1)$ and $(q_0, p_0, q_1)$
If $P_i = P_{i-1}$, either $Q_i$ or $Q_i/2$ is a nontrivial factor of $N$.
If $p_i = p_{i-1}$, either $q_i$ or $q_i/2$ is a nontrivial factor of $N$.
```

Figure 3.2: SQUFOF with Composition

Note that it is faster to approximate roughly which quadratic forms are needed to find the symmetry point than to actually calculate the distance exactly. The result is that after this composition, the quadratic form is close to the symmetry point but not exactly on it. In the last **while** loop, the search for the symmetry point has to be made in both directions, but this search is typically very short. The only forms that need to be stored are those with an index that is a power of two. The predictions of which squares provide a non-trivial factor is unnecessary since with Fast Return it is faster to test each square found.

For example, if a square were found on step 56, there should be roughly 28 steps backward in the second phase. If after taking the square root and reversing the direction, F_0 is composed with the quadratic form with index 16, the form with index 8, and the form with index 4, the result will be very close to the symmetry point.

3.3 After Shanks

Shanks died in 1996 leaving several papers on Square Forms Factorization in unpublished form [27]. These were incomplete and contained very few proofs. However, Shanks suggests he had proofs. In his main paper, for example, he states that SQUFOF has an average runtime of $O(\sqrt[4]{N})$ and says this will be proven later. Since the paper was not complete, it may never be known whether or not Shanks possessed a proof of this critical detail.

Although Shanks did not publish his work, it has been picked up by several people. Williams [26], in 1985, provides a description of SQUFOF based on ideals and lattices. Williams demonstrates how continued fractions can be seen as an intuitive extension of the theory of lattice minima (§4) and uses lattices to connect composition with distance (§6). Williams went further by generalizing these results complex cubic fields. Riesel [18], also in 1985, gives a good exposition of SQUFOF using the theory of continued fractions. Buell [2], in 1989, provides a thorough analysis of quadratic forms and gives a description of SQUFOF.

Recently, the thesis of Jason Gower in [9] discussed the runtime analysis of SQUFOF and Shanks' belief that the expected runtime is $O(\sqrt[4]{N}) = L(1, 1/4)$. Jason Gower provides a good description of continued fractions, quadratic forms, and the basic SQUFOF algorithm. He also did some research and experiments on the use of a multiplier and discovered that it could be an effective method to accelerate the algorithm. His analysis of the runtimes is based on the distribution of square forms through the ambiguous cycles and is supported by extensive data. This work was completed in December of 2004.

However, although much of the theory about SQUFOF was known, some of it was not and most of it was very scattered with few proofs. Until now, no complete explanation of why SQUFOF works had been published. Therefore, in the appendices I have compiled all of this information and provided proofs and examples. I first analyzed continued fractions, showing that taking the conjugate of the numerator provide a new and intuitive explanation of the reversal of direction (Lemmas A.1.3 - A.1.4). This understanding then provides immediate results about the periodicity and symmetries of the cycles (Lemmas A.1.5 - A.1.7). These symmetry points correspond to symmetry points in the quadratic form cycle (Lemma A.2.6).

The fact that the square of any ambiguous cycle is the principal cycle is of fundamental importance to SQUFOF and is proved in Theorem A.2.8. This result was probably understood by Shanks, but apparently never stated explicitly. This fact implies that after taking the square root of a quadratic form in the principal cycle the result is in an ambiguous cycle so that there will be a symmetry point.

The application of ideals to quadratic forms required a slight generalization to the standard description of ideals (Lemma A.3.2). This slight extra generality allows a proof that multiplication of ideals corresponds to composition of quadratic forms. The well-known result about the multiplication of ideals was also generalized slightly (A.3.3) so that the case of quadratic forms with discriminant $\equiv 1 \pmod 4$ could be handled without working in $\mathbb{Z}\left[\frac{\sqrt{N}+1}{2}\right]$. The connection between lattice reduction and continued fraction reduction also shows that a distance for reduction may be computed (Lemma A.5.1).

Theorem A.5.2 relates composition of quadratic forms with distance and is a generalization of the distance formula provided by Williams in [26]. This generalization is essential for SQUFOF as it relates distances in different non-principal cycles with composition. Lemma

A.5.3, which relates the entire distance around different cycles, is proved based on this. With these facts, it is possible to prove that the distance traveled in the non-principal cycle before finding a symmetry point will be one half of the distance traveled in the principal cycle to find the square form.

Appendix A.6 provides a complete description of Square Forms Factorization, including an explanation of why Shanks was able to know whether or not a square was proper and an explanation of what Shanks referred to as Fast Return. Fast Return, an idea that is typically not implemented with SQUFOF, uses composition to significantly speed up the second phase of the algorithm.

Chapter 4

Computer Based Experiments

Appendix B includes the C implementation of SQUFOF with Fast Return. These algorithms have also been implemented in Magma. These have both been tested extensively for functionality. The runtimes have also been compared to elliptic curve method, multiple polynomial quadratic sieve, and the number field sieve. Although a thorough comparison of SQUFOF with other algorithms has not been done, it is fairly clear that on a single computer SQUFOF is slower than elliptic curves, the number field sieve, or the multiple polynomial quadratic sieve for most numbers. However, there have been several other areas that have been researched. It has been conjectured by Pomerance [17] that a parallel implementation of SQUFOF would be competitive. This area of research provided some valuable insights into these algorithms.

4.1 Two Parallel Implementations

The current trend in ultra-fast computing is to use a large number of inexpensive processors instead of a single expensive processor [7]. With the large amount of computation required for factorization, the efficiency of a parallel implementation is especially important for these algorithms. In 1982, Shanks and Cohen attempted a parallelization by having multiple processors attempt to factor N, $3N$, $5N$, etc. Gower's recent Ph.D. thesis (under Dr. Wagstaff) [9] analyzed the use of multipliers and found them to be effective in general but didn't provide much evidence on their efficiency for parallelization.

Parallelization by multipliers has been implemented in C using the MPI library for the purpose of runtime comparison and is included in Appendix B.

This research has discovered that composition of quadratic forms provide a different possible approach to parallelization. By computing a quadratic form several steps into the cycle and squaring it a reasonable[1] number of times, a quadratic form far out into the cycle can be found. Call it F. The first processor may be assigned to search this segment for a perfect square. The next processor can search from F to F^2, the next from F^2 to F^3, etc. Each of these segments will be roughly equal sizes. When a processor finds a perfect square, it may use the quadratic forms involved in computing its segment in order to come close to the symmetry point. Specifically, this last step is made easier if the starting point of the segment is found by squaring. In pseudo-code:

```
Given $N$ to be factored:
$r \leftarrow \lfloor\sqrt{N}\rfloor$
$F_0 \leftarrow (1, 2r, N - r^2)$
Cycle $F_0$ several steps forward.
**for** $i = 1$ to size    (size is the logarithmic size of a segment.)
    $F_i \leftarrow F_{i-1} * F_{i-1}$

Processor 0:
    Assign one processor to search from $F_0$ to $F_{size}$.
    $F_{start} \leftarrow F_{size}$
    $F_{end} \leftarrow F_{size}^2$
    $F_{rootS} \leftarrow F_{size-1}$
    $F_{rootE} \leftarrow F_{size}$
    $F_{step} \leftarrow F_{size-1}$
    **while** A factor hasn't been found
        Wait for a processor to be free and send $F_{start}$, $F_{end}$, and $F_{rootS}$.
        $F_{start} \leftarrow F_{end}$
        $F_{rootS} \leftarrow F_{rootE}$
```

[1] Empirically, 20-30 times works well. This parameter isn't critical.

```
    $F_{rootE} \leftarrow F_{rootE} * F_{step}$
    $F_{end} \leftarrow F_{rootE}^2$

Processor $n$:
  Receive $F_{start}$, $F_{end}$, and $F_{rootS}$
  count $\leftarrow$ 0
  $F_0 = (A, B, C)$
  **while** A factor is not found and $F_{start} \neq F_{end}$
    Cycle $F_{start}$ forward 2 steps.
    count $\leftarrow$ count+1
    **if** $A$ is a perfect square
      $F_{test} \leftarrow F_{start}^{-1/2}$
      $F_{test} \leftarrow F_{test} * F_{rootS}$
      **for** $j = size$ to 1                (This loop composes $F_{test}$ with the necessary
        **if** count $> 2^j$                   forms to bring it close to the symmetry point.)
          $F_{test} \leftarrow F_{test} * F_j$
          count $\leftarrow$ count $-2^j$
        Search in both directions from $F_{test}$ for a symmetry point.
        **if** Factorization found at symmetry point, output and quit.
  **if** A factor is still not found
    Receive new $F_{start}$, $F_{end}$, and $F_{rootS}$ and start over.
```

Figure 4.1: Parallel SQUFOF

Since there is no overlap between the segments searched by the processors and since the perfect squares are, apparently, distributed randomly, this parallelization should be extremely efficient. The size of the segments is determined by how far the first **for** loop goes. There are only two hazards when choosing this. If the segment size is too small, the processors will finish their segments so quickly that receiving new segments will become a bottleneck. Alternately, if the segments are too long, the processors may divide up more than the entire cycle, so that there is overlap. However, except for rare numbers that will factor trivially fast regardless, there is significant room in between these two bounds.

This parallel version has been implemented in C using the MPI library. The code is included in Appendix B.

There are several characteristics of these algorithms that were analyzed in this comparison. The test integers were all products of randomly chosen primes of roughly equal size generated by Magma. 80 bit, 100 bit and 120 bit integers were all tested on 20, 30, 40 and 50 processors. This allows an analysis of both how each algorithm is affected by the size of the integers and how efficiently each algorithm uses an increasing number of processors.

The first comparison is an analysis of which algorithm was faster on a larger number of the integers. Table 4.1 shows these results.

		Processors			
		20	30	40	50
bits	80	0.39	0.38	0.39	0.30
	100	0.37	0.34	0.43	0.40
	120	0.35	0.31	0.29	0.39

Table 4.1: % Wins by the Segments Parallelization

The data is clearly showing that parallelization with multipliers is faster than segments and shows no apparent pattern as to how the size of the integers or the number of the processors might affect this.

The next comparison is an analysis of the average runtimes. This comparison, demonstrated in Table 4.2 is a bit more interesting.

		Segments			Multipliers		
	Processors:	80	100	120	80	100	120
Bits	20	5.04	157.27	5416.97	2.95	82.62	3255.65
	30	3.03	116.76	4287.37	1.95	59.75	2028.72
	40	2.53	74.50	2690.76	1.68	52.90	1683.57
	50	2.13	77.67	2538.80	1.23	53.60	1287.80

Table 4.2: Comparison of Average Runtimes

A quick survey of this data shows that for the segments parallelization, the average runtime was cut in half from 20 processors to 40, while the multipliers implementation didn't do quite so well. This can be represented by multiplying each runtime by the number of processors and then dividing by the mean runtime for that many bits to place all the results on a single graph:

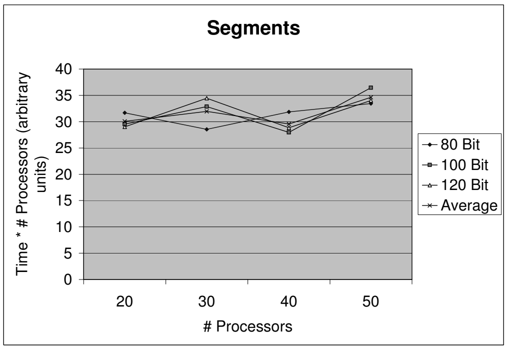

Figure 4.2: Segments Efficiency of Parallelization

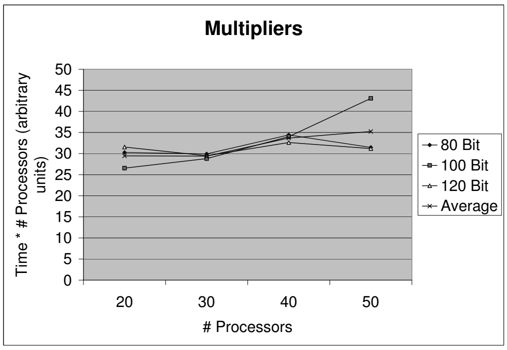

Figure 4.3: Multipliers Efficiency of Parallelization

These graphs show that the efficient use of multiple processors for the segments parallelization is roughly unaffected by increasing the number of processors, while the multipliers parallelization is less efficient at using a larger number of processors. This is the expected result. As Jason Gower demonstrated in [9], the use of a multiplier can decrease the average runtime by an average of 27%. Therefore, for small numbers of processors, using multipliers should immediately cut the average runtime down. However, for larger numbers of processors, the multipliers available aren't used as efficiently.

This same trend can be expressed by graphing the ratios of average runtimes:

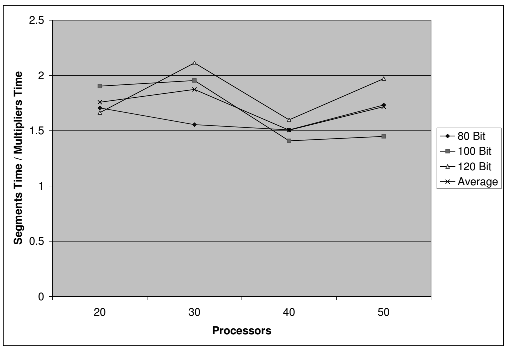

Figure 4.4: Ratios of Segments v. Multipliers

Although the data isn't completely clear, the trend is toward segments being faster than multipliers if enough processors are used. Based on the ratios, a linear regression predicts a crossover at 180 processors. However, as larger multipliers are used, the efficiency of the multipliers should drop off significantly, so the actual crossover will be earlier than that.

Considering that directly, SQUFOF probably has average runtime of $O(\sqrt[4]{N})$, and it appears to use the processors available with near perfect efficiency, the expected runtime on multiple processors is $O(\sqrt[4]{N}/np)$, where np is the number of processors. Based on the data that for 120 bits, the factorization took an average of 31.91 processor hours, it is then expected that RSA-576, the last RSA challenge number factored, would take $31.91 * 2^{(576-120)/4} = 6.63 * 10^{35}$ processor hours. Thus, for example, with 10^{20} processors, it would take $6.63 * 10^15$ hours $= 7.57 * 10^{11}$ years. Both of these figures are extremely unfeasible. Therefore, a parallel implementation of SQUFOF will probably never be competitive with other algorithms.

4.2 Test of Direction

Based on the symmetry of the cycles of quadratic forms, any fast test to determine whether or not the continued fraction expansion is in the correct direction, that is, whether or not it has passed the factorization yet, would provide a faster factorization algorithm by performing a binary search (Figure 4.5).

```
Given $N$ to be factored:
$Q_0 \leftarrow 1$, $P_0 \leftarrow \lfloor\sqrt{N}\rfloor$
Apply equation (2.2) for 2 steps.
$F_0 \leftarrow (Q_2, 2 \cdot P_2, -Q_3)$
$i \leftarrow 0$
while $F_i$ is in the right direction
    $F_{i+1} \leftarrow F_i * F_i$
    $i \leftarrow i + 1$
$F_{last} \leftarrow F_{i-1}$
while $i \geq 0$
    if $F_{last} * F_i$ is in the right direction
        $F_{last} \leftarrow F_{last} * F_i$
    $i \leftarrow i - 1$
Search forward from $F_{last}$ (should be only a couple steps)
        to obtain the factorization.
```

Figure 4.5: Binary Search based on a Test of Direction

Example 8. $N = 2035153$, $Q_0 = 1$, $P_0 = 1426$

Then, $F_0 = (1, 2 \cdot 1426, -1677)$. The square of this form would be itself, so we recursively step forward a few steps to obtain: $F_8 = (663, 2774, -168)$.

This form is squared until it is past the symmetry point:

$$F_8 * F_8 = (1569, 1522, -928) = F_{22}$$

$$F_{22} * F_{22} = (896, 2798, -87) = F_{44}$$

$$F_{44} * F_{44} = (1648, 1726, -783) = F_{82}$$

$$F_{82} * F_{82} = (411, 2104, -2259) \text{ is past the symmetry point.}$$

$$F_{82} * F_{44} = (9, 2846, -1136) = F_{134}$$

$$F_{134} * F_{22} \text{ is past the symmetry point.}$$

$$F_{134} * F_{8} = (1153, 1898, -984) = F_{144}$$

Recursively stepping forward from here, it is quickly found that $F_{147} = (-1008, 2018, 1009)$ and $F_{148} = (1009, 2018, -1008)$. Therefore, 1009 is the symmetry point and $2035153 = 1009 \cdot 2017$.

The decisions for this example of whether or not a form was reversed (past the symmetry point) were determined by merely comparing with the actual continued fraction expansion. However, ideally this could be done without doing the entire expansion. The function that determines whether or not a form is reversed is called a *test of direction.* There were several possible candidates:

Conjecture 1. If $Q_i | Q_{i-1}$, $(Q_i)^3 \nmid Q_{i-1}$, and Q_i is not a power of 2, then the continued fraction expansion is in the correct direction.

In all other cases, this test provides no information. The condition $Q_i | Q_{i-1}$ does not occur very frequently, so on its own it cannot provide a useful test of direction. However, the attempt to explain this empirical pattern has provided several possible alternatives:

First, it is possible to find a multiple k such that in the expansion with Q_i unchanged, P_i replaced by kP_i and N replaced by k^2N, this condition is met. For notation, let $Q_i = Q'_0$. Since using $-k$ instead of k produces the same sequence in the opposite direction, it is necessary to relate this sequence to the original sequence in order to extract useful information. From this, it has so far been found that if $Q_{i+3} | Q'_1$ and $Q'_0 | Q'_{-1}$, then the original expansion is in the correct direction. Finding multiples such that the second condition is satisfied can be accomplished by using continued fractions to find the convergents of $\frac{\sqrt{N} - R_i}{Q_i^2}$, where $R_i \equiv \sqrt{N} \equiv P_i - \frac{P_i^2 - N}{2P_i} \pmod{Q_i^2}$.

This provides a test of direction that returns a result about 2% of the time. This would potentially provide a polynomial time factorization algorithm, except that the time required to do this test of direction increases with the size of N.

One other possible test of direction involves *reduction distance.* After a quadratic form is squared, it often requires multiple steps in order to reduce to a quadratic form within the standard bounds. This is referred to as reduction distance. Since the overall distance from a symmetry point is doubled when a quadratic form is squared, it makes some intuitive sense that this distance from the nearest symmetry point should be related to this reduction distance. Empirically, the reduction distance for a quadratic form oriented away from the nearest symmetry point tends to be shorter. Concerning the conjecture, if $Q_i \mid Q_{i-1}$, then the square of $(Q_i, P_{i-1}, -Q_{i-1})$ is $(Q_i^2, P_{i-1}, -Q_{i-1}/Q_i)$ and the reduction distance is 0, so this would provide an explanation for Conjecture 1.

Empirically, comparing reduction distances is a test of direction that generally works near the symmetry point, but is inconclusive away from the symmetry point. Thus, one other idea may be necessary in order to apply this. Using the *class number formula* [3], it is possible to approximate what the distance to the symmetry point should be. This approximation may be able to come close enough to the symmetry point that some combination of the two tests of direction may be able to provide the necessary information.

4.3 Fast Numbers

The other area of research was whether there were some integers that would factor extremely fast and whether these would pose a security risk. In general, it seems likely that the numbers that factor extremely fast are of the form $(f^2m + c)(g^2m + d)$, where m is "large" and a,b,c, and d are relatively small. The general conditions on these variables appear to be very complicated. The simpler special case of $p(p + 2a)$, for which it is fairly easy to analyze the conditions exactly and determine how often this case occurs, provides some indication of the complexity of this question. Although an exact solution may be developed for any particular case, for the general case it has so far only been possible to approximate how often these cases occur.

Proposition 1. Let $N = p(p+2a)$, where p and a are integers with $0 < a < \sqrt{p}$. Then the Square Forms Factorization algorithm will provide a factorization of N on the second step with runtime $O(\log N)$. Specifically, $Q_2 = a^2$ and $Q_1 \neq a$.

Proof: Let $Q_0 = 1$.

$$(a-1)^2 < a^2 < p < 2p$$

$$(a-1)^2 - 2p < 0$$

$$p^2 + 2ap + (a-1)^2 - 2p < p^2 + 2ap < p^2 + 2ap + a^2$$

$$(p+a-1)^2 < N < (p+a)^2$$

$$\lfloor \sqrt{N} \rfloor = p + a - 1$$

Therefore, $P_0 = p + a - 1$ and $Q_1 = N - P_0^2 = 2p - (a-1)^2$. $b_1 = \left\lfloor \frac{2p+2a-2}{2p-(a-1)^2} \right\rfloor$. Now,

$$a^2 - a - p < 0$$

$$p + a - 1 < 2p - (a-1)^2$$

$$\tfrac{2p+2a-2}{2p-(a-1)^2} < 2$$

$$b_1 = 1$$

Therefore, $P_1 = p - a(a-1)$. $Q_2 = \frac{N-P_1^2}{Q_1} = \frac{2a^2p-a^2(a-1)^2}{2p-(a-1)^2} = a^2$.

If $Q_1 = a$, then $2p - (a-1)^2 = a$, so that $a = \frac{1 \pm \sqrt{8p-3}}{2}$ and $8p - 3 \equiv 5 \pmod 8$ and is thus not a perfect square. Since a is an integer, this is then impossible.

$$y_0 = \frac{\sqrt{N} - p + a(a-1)}{a} = (a-1) + \frac{\sqrt{N} - p}{a}$$

$$y_1 = \frac{a}{\sqrt{N} - p} = \frac{\sqrt{N} + p}{2p} = 1 + \frac{\sqrt{N} - p}{2p}$$

The factor of p is thus found extremely rapidly. Since the only part of the algorithm that takes a significant amount of time is approximating the square root of N, the algorithm has a runtime of $O(\log N)$ for numbers that meet this criteria. **QED**

For the other cases, the fact that $p(4p + 4a + 1)$ and $p(4p + 4a + 3)$ have significantly different bounds for a should be some indication as to how quickly this problem can get messy. However, some approximations can still be determined empirically. Let $N = (f^2m + c)(g^2m + d)$. Then empirically N factors quickly for approximately the same cases as when $f^2g^2N = (f^2g^2m + g^2c) * (f^2g^2m + f^2d)$ factors quickly. Let $p = f^2g^2m + g^2c)$ and let $a = (f^2d - g^2c)/2$ or $(f^2d - g^2c - 1)/2$, then $f^2g^2N = p(p + 2a)$ or $p(p + 2a + 1$, respectively. In either case, the two factors $(f^2g^2m + g^2c)$ and $(f^2g^2m + f^2d)$ must have roughly half their bits the same.

For a given $N = qr$, consider the probability that there exist f and g such that g^2q and f^2r have the same number of bits and the first 3 bits are the same. This probability can easily be tested empirically. The continued fraction expansion for q/r provides potential values for f and g. A test of 10^{15} pairs of 100 bit integers and 10^{15} pairs of 300 bit primes show that this happens approximately 11% of the time in each case, with only a couple of pairs that had multiple values of f and g meeting the criteria. It seems reasonable to assume that if the size of the paired integers wasn't required to be equal, the probability would be different but would still be fairly constant. Let this constant be k. Then for each bit after that, there is a 50% probability of q and r having the same value. Since roughly the first half of the bits must be equal, the probability is $k * (.5)^{\log_2 N/2-3} = 8k/\sqrt{N} = O(1/\sqrt{N})$. Thus, these numbers, although interesting, are extremely rare and should not be a concern in normal operations.

Chapter 5

Conclusions and Future Work

This report has described and explained why Square Forms Factorization works, completing the work started by Daniel Shanks. Specifically, a thorough correspondence was described between quadratic forms, continued fractions, ideals and lattices, resulting in a proof of a generalized distance formula. This allowed an explanation of how Square Forms Factorization works. Several variations have been considered. An analysis shows that the use of multipliers decreases the average runtime by 25%. However, the use of multipliers alone is an inefficient method of implementing Square Forms Factorization on a large number of processors. Parallelization by dividing the cycle into segments for each processor appears to be more efficient with larger number of processors. The crossover between these two methods appears to be at about 100 processors. In light of this, the best parallelization would probably involve some combination. Two alternatives were considered but not tried. It would be fairly simple to use a single multiplier and then use segments to parallelize. It may be even more efficient to use several multipliers and then use segments to divide each of those between many processors.

With the parallelization, it is important to realize that these parallelizations are all very easy to implement and would be very effective as a distributive process. The only information that the master processor needs to send is basic information needed to assign a segment. The other processor then works independently until it either finds a factor or finishes its segment.

Another alternative considered using a test of direction to perform a binary search. However, the test of direction only appeared to be effective near the symmetry points and even with an approximation of the regulator it was not possible to get close enough to the symmetry points for the test of direction to be useful.

One other property of Square Forms Factorization that was considered is which numbers factor extremely fast. Although there are integers of every size that factor in time $O(\log N)$, they are rare. The result was that an integer N has a probability $O(1/\sqrt{N})$ of meeting this criteria. Therefore, this isn't a threat worth considering seriously.

Chapter 6

Glossary

Conjugate: modification to an algebraic number changes the sign of a certain term in an expression, traditionally the part of an expression that is either irrational or imaginary. In the context of continued fractions, it applies to changing the sign of a **residue**, the integer part of the numerator of an x_i in the continued fraction expansion.

Cryptography: the process of encoding and decoding information to prevent it from being read by anyone who intercepts the message. Used in variety of civil and military applications.

Euclid's Algorithm: a fast algorithm for determining the greatest common divisor of two integers. Given x and y, the extended Euclidian algorithm also determines the coefficients a and b such that $ax + by = \gcd(x, y)$.

Floor: the greatest integer less than or equal to a given number. Symbol: $\lfloor x \rfloor$

Greatest Common Divisor: largest integer that divides a pair or group of integers. Symbol: $gcd(x, y)$

Modular Arithmetic: two integers are considered equal (congruent) if their difference is divisible by a base. Thus, $3 \equiv 10 \pmod 7$. Numbers are represented by integers between 0 and $N-1$, where N is the base. Multiplication, addition, and subtraction are normal, except that the results are reduced to the range 0 to $N-1$. Division is performed by reducing fractions to least terms, applying an extended Euclid's algorithm to find the inverse of the denominator, and then performing multiplication. Symbol: (mod N)

Modulo: operation related to division that returns the remainder:

$\frac{73}{11} = 6\frac{7}{11}$, so 73 modulo 11 = 7. Symbol: % or (mod N).

Non-square integer that is not a perfect square of another integer.

NUCOMP-NUDUPL: algorithms designed by Daniel Shanks to perform composition of quadratic forms quickly.

Parallelization: the distribution of a single task between multiple processors working simultaneously.

Perfect Square: integer that is the square of another integer. Thus, 9 is a perfect square because $3^2 = 9$.

Prime: integer that has no divisors other than itself and 1. The first several are $2, 3, 5, 7, 11....$

Primitive: a characteristic that implies not having common factors. For quadratic forms, a form whose coefficients do not contain a common factor. For ideals, an ideal whose elements does not contain a common integer factor.

Pseudo-square: the denominator of an x_i in the continued fraction expansion, denoted Q_i. When $Q_0 = 1$, $(-1)^i Q_i$ is a quadratic residue and in general $-Q_iQ_{i-1}$ is a quadratic residue.

Quadratic Reciprocity Law (Gauss): determines which numbers are quadratic residues of a prime:

Symbol: $\left(\frac{a}{p}\right) = 1$ if $x^2 \equiv a \pmod p$ has a solution, -1 if it does not, and 0 if $p \mid a$.

Theorem: For p and q distinct primes:

$$\left(\frac{p}{q}\right)(\frac{q}{p}) = (-1)^{(p-1)(q-1)/4}$$

$$\left(\frac{2}{p}\right) = (-1)^{(p^2-1)/8}$$

$$\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2}$$

Quadratic Residue: a perfect square modulo some base N. 2 is a quadratic residue of 7 because $3^2 \equiv 2 \pmod 7$.

Relatively Prime: a description of two integers that have no common divisors. Thus, 8 and 15 are relatively prime, even though they are not prime by the normal definition.

Residue: an integer that remains in the numerator after b_i has been subtracted from x_i in the continued fraction expansion.

RSA: a cryptology algorithm named after Rivest, Shamir, and Adleman. It was earlier developed by Clifford Cooks of GCHQ, but this was only recently declassified. Its security is dependent on the difficulty of factorization [5].

SQUFOF: Square Forms Factorization algorithm, developed by Daniel Shanks in 1975.

Index

Bibliography

[1] W. R. Alford, A. Granville, and C. Pomerance. There are infinitely many carmichael numbers. In *Annals of Math*, volume 139, pages 703–722. 1994.

[2] D. A. Buell. *Binary Quadratic Forms: Classical Theory and Modern Computations.* New York: Springer-Verlag, 1989.

[3] Harold Davenport. *Multiplicative Number Theory.* New York: Springer-Verlag, 1980.

[4] B. N. Delone and D. K. Faddeev. *The Theory of Irrationalites of the Third Degree.* American Mathematical Society, 1964.

[5] J. H. Ellis. The history of non-secret encryption. *Cryptologia*, 1999.

[6] A. K. Lenstra et all. The development of the number field sieve. In *Lecture Notes in Mathematics*, number 1554. Berlin: Springer-Verlag, 1993.

[7] David J. Kuck et all. Parallel supercomputing today and the cedar approach. *Science*, 231(4741):967–974, 1986.

[8] Carl Friedrich Gauss. *Disquisitiones Arithmeticae.* New Haven: Yale University Press, 1966. Trans. by Arthur A. Clarke.

[9] J. E. Gower. *Square Form Factorization.* PhD thesis, Purdue University, December 2004.

[10] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers.* Oxford: Clarendon Press, 1979.

[11] D. H. Lehmer and R. E. Powers. On factoring large numbers. *Bull. Amer. Math. Soc*, 37:770–776, 1931.

[12] H. W. Jr. Lenstra. On the calculation of regulators and class numbers of quadratic fields. volume 56, pages 123–150. London Mathematical Society, 1980.

[13] H. W. Jr. Lenstra. Factoring integers with elliptic curves. In *Annals of Mathemtaics*, volume 126, pages 649–673. 1987.

[14] R. A. Mollin and A. J. van der Poorten. A note on symmetry and ambiguity. *Bull. Austral. Math. Soc.*, 1995.

[15] Michael A. Morrison and John Brillhart. A method of factoring and the factorization of f_7. *Mathematics of Computation*, 29(129):183–205, January 1975.

[16] C. Pomerance. A tale of two sieves. *Notices American Mathematical Society*, (43):1473–1485, 1996.

[17] C. Pomerance. Personal Communication to David Joyner, June 2004.

[18] Hans Riesel. *Prime Numbers and Computer Methods for Factorization.* Boston : Birkhuser, 1985.

[19] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[20] Daniel Shanks. The infrastructure of a real quadratic field and its applications. Number Theory Conference: University of Colorado, Boulder, Colorado, August 1972.

[21] Daniel Shanks. On gauss and composition ii. *Number Theory and Applications*, pages 179–204, 1989.

[22] Daniel Shanks. Analysis and improvement of the continued fraction method of factorization. Transcribed by Stephen McMath March 2004., circa 1975.

[23] Daniel Shanks. Squfof notes. Transcribed by Stephen McMath March 2004., circa 1975.

[24] Wade Trappe and Lawrence C. Washington. *Introduction to Cryptography with Coding Theory.* New Jersey: Prentice-Hall, Inc, 2002.

[25] W. Sartorius von Waltershausen. *Gauss: A Memorial.* Colorado Springs, Colorado, 1966.

[26] Hugh C Williams. Continued fractions and number-theoretic computations. *Rocky Mountain Journal of Mathemtaics*, 15(2), 1985.

[27] Hugh C Williams. Daniel shanks (1917-1996). *Mathematics of Computation*, 66(219):929–934, July 1997.

Appendix A

Proofs

The largest part of this research has been developing formal proofs of the concepts supporting Square Forms Factorization. Section A.1 introduces and proves a number of results pertaining to continued fractions. Section A.2 introduces and proves several results about binary quadratic forms and describes their connection to continued fractions. Section A.3 analyzes how ideals are related to quadratic forms and section A.4 analyzes how lattices are closely tied in with ideals. Section A.5 uses the connection between quadratic forms and lattices to prove the general distance formula, which is used in section A.6 to explain why SQUFOF works.

A.1 Continued Fractions

One tool used by many different algorithms is the continued fraction expression for $\sqrt{N}$, where N is the number to be factored. This expression is calculated recursively[10]:

$$x_0 = \sqrt{N},\ b_0 = \lfloor x_0 \rfloor \tag{A.1}$$

$$\forall i \geq 1\ x_i = \frac{1}{x_{i-1} - b_{i-1}},\ b_i = \lfloor x_i \rfloor \tag{A.2}$$

$$\sqrt{N} = b_0 + \frac{1}{b_1 + \frac{1}{b_2 + \ldots}} \tag{A.3}$$

Observe that solving equation (A.2) for x_{i-1} gives $x_{i-1} = b_{i-1} + \frac{1}{x_i}$. Repeatedly substituting this into itself gives equation (A.3).

Before developing the theory too much further, allow me to offer one example of how this works, just so that the pieces make sense to you. To simplify the expansion some, the integers taken out in the third step of each line are the b_i:

Example 9.

$$\begin{array}{rr}
x_0 = \sqrt{41},\ b_0 = 6 & \sqrt{41} = 6 + \frac{1}{x_1} \\
x_1 = \frac{1}{\sqrt{41}-6} = \frac{\sqrt{41}+6}{5} = 2 + \frac{\sqrt{41}-4}{5} & \sqrt{41} = 6 + \frac{1}{2+\frac{1}{x_2}} \\
x_2 = \frac{5}{\sqrt{41}-4} = \frac{\sqrt{41}+4}{5} = 2 + \frac{\sqrt{41}-6}{5} & \sqrt{41} = 6 + \frac{1}{2+\frac{1}{2+\frac{1}{x_3}}} \\
x_3 = \frac{5}{\sqrt{41}-6} = \frac{\sqrt{41}+6}{1} = 12 + \frac{\sqrt{41}-6}{1} & \sqrt{41} = 6 + \frac{1}{2+\frac{1}{2+\frac{1}{12+\frac{1}{x_4}}}}
\end{array}$$

From this step, it is evident that $x_4 = x_1$ and the cycle repeats from here. Observe that if the sequence starts with $x_0 = \sqrt{41} + 6$, then also $x_3 = x_0$. Regardless, the expression on the right, if truncated at any point provides a rational approximation to $\sqrt{41}$. Often this will be written as merely $[6, 2, 2, 12, ...]$ to save space or $[6, \overline{2, 2, 12}]$ to indicate that this part repeats. The various numbers on the left have some important properties that we will now analyze in some depth.

Throughout, assume that N is an odd positive integer and is not a perfect square. For number theory purposes, let

$$x_0 = \frac{\sqrt{N} + P_{-1}}{Q_0}$$

where P_{-1}, Q_0 are integers chosen such that

$$P_{-1}^2 \equiv N \pmod{Q_0}, \quad 0 < P_{-1} < \sqrt{N}, \quad \text{and } |\sqrt{N} - Q_0| < P_{-1}. \tag{A.4}$$

There are many ways of doing this[1]. The recursive formulas are:

[1] Choosing $x_0 = \sqrt{N} + \lfloor\sqrt{N}\rfloor$, so that $P_{-1} = \lfloor\sqrt{N}\rfloor$ and $Q_0 = 1$ is one possibility. Choosing $x_0 = \frac{\sqrt{N}+P_{-1}}{2}$, where $P_{-1} = \lfloor\sqrt{N}\rfloor$ or $\lfloor\sqrt{N}\rfloor - 1$, such that P_{-1} is odd, is another possibility.

$$x_{i+1} = \frac{1}{x_i - b_i} \quad b_i = \lfloor x_i \rfloor,\ i \geq 0$$

Formally, the assumed equation is:

$$x_{i+1} = \frac{Q_i}{\sqrt{N} - P_i} = \frac{\sqrt{N} + P_i}{Q_{i+1}} = b_{i+1} + \frac{\sqrt{N} - P_{i+1}}{Q_{i+1}},\ i \geq 0 \tag{A.5}$$

Note that this equation serves as a definition of $Q_i, P_i, Q_{i+1}, P_{i+1} \in \mathbb{Q}$, so that these equations are true regardless of the conditions on these variables. Theorem A.1.1 provides some well-known fundamental properties and identities of continued fractions. In [18], Hans Riesel provides very clear proofs of most of this.

Theorem A.1.1. *[18] In the continued fraction expansion of x_0 satisfying (A.4), each x_i reduces to the form $\frac{\sqrt{N}+P_{i-1}}{Q_i}$, with (a) $N = P_i^2 + Q_iQ_{i+1}$, (b) $P_i = b_iQ_i - P_{i-1}$, (c) $b_i = \left\lfloor \frac{\lfloor\sqrt{N}\rfloor+P_{i-1}}{Q_i} \right\rfloor \geq 1$, (d) $0 < P_i < \sqrt{N}$, (e) $|\sqrt{N} - Q_i| < P_{i-1}$, (f) Q_i is an integer, and (g) $Q_{i+1} = Q_{i-1} + b_i(P_{i-1} - P_i)$. Furthermore, (h) this sequence is eventually periodic.*

Proof:

(a) From (A.5), the equation $\frac{Q_i}{\sqrt{N}-P_i} = \frac{\sqrt{N}+P_i}{Q_{i+1}}$ requires that $N = P_i^2 + Q_iQ_{i+1}$.

(b) It is evident from simplifying the expression on the far right of (A.5) that

$$\frac{\sqrt{N} + P_i}{Q_{i+1}} = \frac{\sqrt{N} + b_{i+1}Q_{i+1} - P_{i+1}}{Q_{i+1}}.$$

Therefore, $P_{i+1} = b_{i+1}Q_{i+1} - P_i$.

(c) For $i = 0$, by the assumption $|\sqrt{N} - Q_0| < P_{-1}$

$$Q_0 < \sqrt{N} + P_{-1}$$

Therefore,

$$b_0 = \left\lfloor \frac{\sqrt{N} + P_{-1}}{Q_0} \right\rfloor \geq 1$$

For $i > 0$, $b_{i-1} = \lfloor x_{i-1} \rfloor$. By the definition of floor, $x_{i-1} - 1 < b_{i-1} \leq x_{i-1}$. If $b_{i-1} = x_{i-1}$, then the continued fraction $[b_0, b_1, ...b_{i-1}]$ is rational and is equal to $x_0 = \frac{\sqrt{N}+P_{-1}}{Q_0}$, which is irrational since N is not a perfect square. Therefore, $x_{i-1} - 1 < b_{i-1} < x_{i-1}$, so that $0 < x_{i-1} - b_{i-1} < 1$. Therefore, $x_i = \frac{1}{x_{i-1} - b_{i-1}} > 1$, so that $b_i = \lfloor x_i \rfloor \geq 1$.

Note that there is no integer between $\lfloor\sqrt{N}\rfloor + P_{i-1}$ and $\sqrt{N} + P_{i-1}$, so it is trivial that $\left\lfloor \frac{\sqrt{N}+P_{i-1}}{Q_i} \right\rfloor = \left\lfloor \frac{\lfloor\sqrt{N}\rfloor+P_{i-1}}{Q_i} \right\rfloor$.

(d-e) The statements $|\sqrt{N} - Q_i| < P_{i-1}$ and $0 < P_{i-1} < \sqrt{N}$ may be proven inductively.

Base case: $i = 1$

$P_0 = \lfloor\sqrt{N}\rfloor$ or $\lfloor\sqrt{N}\rfloor - 1$, so by definition $0 < P_0 < \sqrt{N}$.

Since x_0 meets (A.4), $|\sqrt{N} - Q_0| < P_{-1}$.

Induction: Assume $|\sqrt{N} - Q_i| < P_{i-1}$ and $0 < P_{i-1} < \sqrt{N}$.

Note that these assumptions require that $0 < Q_i < 2\sqrt{N}$. From (c), $0 < x_i - b_i < 1$ means $0 < \frac{\sqrt{N}-P_i}{Q_i} < 1$. Since $Q_i > 0$, $0 < \sqrt{N} - P_i < Q_i$. From the left side of this, $P_i < \sqrt{N}$. Now, either $Q_i \leq \sqrt{N}$ or $Q_i > \sqrt{N}$.

Case 1: If $Q_i \leq \sqrt{N}$, then $\sqrt{N} - P_i < Q_i \leq \sqrt{N}$, so that $P_i > 0$.

Case 2: If $Q_i > \sqrt{N}$, then by (b), $P_i = b_i Q_i - P_{i-1} > b_i\sqrt{N} - \sqrt{N} = (b_i - 1)\sqrt{N} \geq 0$.

Therefore, $0 < P_i < \sqrt{N}$.

Since $x_{i+1} > 1$, it is trivial that $Q_{i+1} < \sqrt{N} + P_i$ so that showing $|\sqrt{N} - Q_{i+1}| < P_i$ reduces to showing $Q_{i+1} > \sqrt{N} - P_i$. Since $1 = \frac{N-P_i^2}{Q_i Q_{i+1}} = \frac{\sqrt{N}+P_i}{Q_i}\frac{\sqrt{N}-P_i}{Q_{i+1}}$, this is equivalent to showing:

$$\frac{\sqrt{N}+P_i}{Q_i} > 1. \tag{A.6}$$

Assume the contrary, that $Q_i \geq \sqrt{N} + P_i$. Then,

$$b_i(\sqrt{N} + P_i) - P_i \leq b_i Q_i - P_i = P_{i-1} < \sqrt{N},$$

$$b_i\sqrt{N} + P_i(b_i - 1) < \sqrt{N},$$

$$\sqrt{N}(b_i - 1) + P_i(b_i - 1) < 0,$$

$$(b_i - 1)(\sqrt{N} + P_i) < 0.$$

But $\sqrt{N}$ and P_i are positive, so this implies $b_i < 1$, contradicting Theorem A.1.1 (c). Therefore, (A.6) holds.

(f) The fact that $N = P_i^2 + Q_iQ_{i+1}$ requires that $Q_{i+1} = \frac{N-P_i^2}{Q_i}$. In order to show that $\forall i$ Q_i is an integer, the statements that Q_i is an integer and $Q_i \mid N - P_i^2$ may be proven inductively.

Base case: $i = 0$

By definition, Q_0 is an integer and $P_{-1}^2 \equiv N \pmod{Q_0}$. But $P_0 \equiv P_{-1} \pmod{Q_0}$, so $P_0^2 \equiv N \pmod{Q_0}$. Therefore, $Q_0 \mid N - P_0^2$.

Induction: Assume for some i, Q_i is an integer and $Q_i \mid (N - P_i^2)$. Then, since $N = P_i^2 + Q_iQ_{i+1}$, $Q_{i+1} = \frac{N-P_i^2}{Q_i}$, so that since $Q_i \mid (N - P_i^2)$, Q_{i+1} is an integer. Also, $Q_i = \frac{N-P_i^2}{Q_{i+1}}$, so that since Q_i is an integer, $Q_{i+1} \mid (N - P_i^2)$, so that $P_i^2 \equiv N \pmod{Q_{i+1}}$. Since $P_{i+1} = b_{i+1}Q_{i+1} - P_i$, $P_{i+1} \equiv P_i \pmod{Q_{i+1}}$. Therefore, $P_{i+1}^2 \equiv N \pmod{Q_{i+1}}$, so that $Q_{i+1} \mid (N - P_{i+1}^2)$ and the induction is complete.

(g) Solving (b) for b_i gives $\frac{P_{i-1}+P_i}{Q_i} = b_i$. Multiply by $(P_{i-1} - P_i)$ to obtain:

$$\frac{P_{i-1}^2 - P_i^2}{Q_i} = b_i(P_{i-1} - P_i)$$

Rearranging and adding $\frac{N}{Q_i}$ gives:

$$\frac{N - P_i^2}{Q_i} = \frac{N - P_{i-1}^2}{Q_i} + b_i(P_{i-1} - P_i)$$

$$Q_{i+1} = Q_{i-1} + b_i(P_{i-1} - P_i)$$

.

(h) Since each x_i and thus the entire sequence that follows it is defined by the two integers Q_i and P_{i-1}, limited by the bounds $0 < Q_i < 2\sqrt{N}$ and $0 < P_i < \sqrt{N}$, there is only a finite number of distinct x_i's. Therefore, for some π and some k, $\forall i \geq k$ $x_i = x_{i+\pi}$. **QED**

The fact that each x_i reduces to the form $\frac{\sqrt{N}+P_{i-1}}{Q_i}$ is important for computational efficiency because this together with (c) imply that floating point arithmetic is not necessary for any of these calculations. Also, by use of (b) and (g), the arithmetic used in this recursion is on integers $< 2\sqrt{N}$.

One application of continued fractions is rational approximations.

$$\sqrt{N} = b_0 + \cfrac{1}{b_1 + \frac{1}{b_2 + ...}}$$

If this continued fraction is truncated at any point, the result is an approximation to $\sqrt{N}$. One might imagine that it is necessary to start simplifying at the lower right end of this expression to obtain this approximation. However, Theorem A.1.2, also included in [18], provides a simpler answer.

Theorem A.1.2. *Let:*

$$A_{-1} = 1,\ A_0 = b_0,\ A_i = b_i A_{i-1} + A_{i-2},\ i > 0$$

$$B_{-1} = 0,\ B_0 = 1,\ B_i = b_i B_{i-1} + B_{i-2},\ i > 0$$

Then for $i \geq 0$, $[b_0, b_1, ...b_i] = \frac{A_i}{B_i}$ and $A_{i-1}^2 - B_{i-1}^2 N = (-1)^i Q_i$.

Note that the last equation gives $A_{i-1}^2 \equiv (-1)^i Q_i \pmod{N}$. Although this equation will change some when generalized to other continued fractions, these denominators $\{Q_i\}$ will consistently be referred to as *pseudo-squares*. A proof of this theorem is given in [18]. Therefore, instead of reproducing the proof, I will provide an example:

Example 10.

$$\begin{aligned}
x_0 &= \sqrt{403} + 20 = 40 + \sqrt{403} - 20 \\
x_1 &= \tfrac{1}{\sqrt{403}-20} = \tfrac{\sqrt{403}+20}{3} = 13 + \tfrac{\sqrt{403}-19}{3} \\
x_2 &= \tfrac{3}{\sqrt{403}-19} = \tfrac{\sqrt{403}+19}{14} = 2 + \tfrac{\sqrt{403}-9}{14} \\
x_3 &= \tfrac{14}{\sqrt{403}-9} = \tfrac{\sqrt{403}+9}{23} = 1 + \tfrac{\sqrt{403}-14}{23} \\
x_4 &= \tfrac{23}{\sqrt{403}-14} = \tfrac{\sqrt{403}+14}{9} = 3 + \tfrac{\sqrt{403}-13}{9} \\
x_5 &= \tfrac{9}{\sqrt{403}-13} = \tfrac{\sqrt{403}+13}{26} = 1 + \tfrac{\sqrt{403}-13}{26} \\
x_6 &= \tfrac{26}{\sqrt{403}-13} = \tfrac{\sqrt{403}+13}{9} = 3 + \tfrac{\sqrt{403}-14}{9} \\
x_7 &= \tfrac{9}{\sqrt{403}-14} = \tfrac{\sqrt{403}+14}{23} = 1 + \tfrac{\sqrt{403}-9}{23}
\end{aligned}$$

From this, a table may be used to recursively calculate the approximation[2]:

[2] Actually, in this case $b_0 = 40$, but since that would be an approximation to $x_0 = \sqrt{403} + 20$, subtracting

i	-1	0	1	2	3	4	5	6
b_i		20	13	2	1	3	1	3
A_i	1	20	261	542	803	2951	3754	14213
B_i	0	1	13	27	40	147	187	708

filling in A_i and B_i from left to right. From the last column, $\sqrt{403} \approx 14213/708$.

Since the continued fraction is eventually periodic, it is reasonable to consider that when it loops around on itself, the terms being considered may have come from some terms "earlier" in the recursion. Example 10 provides some indication as to how the recursive formulas may be reversed, as $\{Q_i\}$ and $\{b_i\}$ are symmetric about x_5, so that after x_5 these numbers are cycled through in reverse order. Lemma A.1.3 addresses how each b_i is calculated two different ways and Lemma A.1.4 shows that by exchanging these two related expressions, the direction is reversed.

Lemma A.1.3.

$$\lfloor \frac{\sqrt{N} + P_i}{Q_i} \rfloor = \lfloor \frac{\sqrt{N} + P_{i-1}}{Q_i} \rfloor = b_i$$

Proof: The second part of this equation, that $\lfloor \frac{\sqrt{N}+P_{i-1}}{Q_i} \rfloor = b_i$ follows from the definition of b_i.

Theorem A.1.1 (e) implies that $Q_i > \sqrt{N} - P_{i-1}$. Therefore,

$$\lfloor \frac{\sqrt{N} + P_i}{Q_i} \rfloor = \lfloor \frac{\sqrt{N} + b_i Q_i - P_{i-1}}{Q_i} \rfloor = b_i + \lfloor \frac{\sqrt{N} - P_{i-1}}{Q_i} \rfloor = b_i. \textbf{ QED}$$

Considering Example 10, it is then natural to suspect that the mechanism for going in the opposite direction will be precisely the same as the standard approach, except that the numerator is changed first. Note that this same change (with the exception of c_0) could be achieved by merely changing the sign of P_{i-1}.

20 from b_0 yields an approximation to $\sqrt{403}$.

Lemma A.1.4. *Let x_i, b_i, P_i, Q_i, and N be as in Theorem A.1.1, $i \geq 0$. Let $y_0 = \frac{\sqrt{N}+P_{i+1}}{Q_{i+1}}$ and let $c_0 = \lfloor y_0 \rfloor$. Define inductively $y_j = \frac{1}{y_{j-1}-c_{j-1}}$. Then $c_0 = b_{i+1}$ and $y_j = \frac{\sqrt{N}+P_{i-j+1}}{Q_{i-j+1}}$, $j \geq 0$.*

Proof: By (A.6) and Lemma 1, $c_0 = \lfloor y_0 \rfloor = \lfloor \frac{\sqrt{N}+P_{i+1}}{Q_{i+1}} \rfloor = b_{i+1}$. By mathematical induction it suffices to prove the case $j = 1$. Using Theorem A.1.1

$$y_1 = \frac{1}{y_0 - c_0} = \frac{1}{\frac{\sqrt{N}+P_{i+1}}{Q_{i+1}} - b_{i+1}} = \frac{1}{\frac{\sqrt{N}+P_{i+1}-b_{i+1}Q_{i+1}}{Q_{i+1}}}$$

$$= \frac{1}{\frac{\sqrt{N}-P_i}{Q_{i+1}}} = \frac{\sqrt{N}+P_i}{\frac{N-P_i^2}{Q_{i+1}}} = \frac{\sqrt{N}+P_i}{Q_i} \quad \textbf{QED}$$

This demonstrates an important fact about continued fractions, the fact that the direction of the sequences of pseudo-squares and residues can be reversed (i.e. the indices decrease) by making a slight change and applying the same recursive mechanism.

Using Lemma A.1.4, x_3 may be used, for example, to find x_2 and x_1. Continuing this process, denote the terms before x_0 as $x_{-1}, x_{-2}, \ldots$. Define Q_{-i} and P_{-i} similarly[3]. Example 11 demonstrates this with the continued fractions from Example 10:

Example 11. $x_3 = \frac{\sqrt{403}+9}{23}$ and $P_3 = 14$, so let $y_0 = \frac{\sqrt{403}+14}{23}$ to obtain

$$\begin{aligned}
y_0 &= \tfrac{\sqrt{403}+14}{23} = 1 + \tfrac{\sqrt{403}-9}{23} \\
y_1 &= \tfrac{23}{\sqrt{403}-9} = \tfrac{\sqrt{403}+9}{14} = 2 + \tfrac{\sqrt{403}-19}{14} \\
y_2 &= \tfrac{14}{\sqrt{403}-19} = \tfrac{\sqrt{403}+19}{3} = 13 + \tfrac{\sqrt{403}-20}{3} \\
y_3 &= \tfrac{3}{\sqrt{403}-20} = \tfrac{\sqrt{403}+20}{1} = 40 + \tfrac{\sqrt{403}-20}{1} \\
y_4 &= \tfrac{1}{\sqrt{403}-20} = \tfrac{\sqrt{403}+20}{3} = 13 + \tfrac{\sqrt{403}-19}{3} \\
y_5 &= \tfrac{3}{\sqrt{403}-19} = \tfrac{\sqrt{403}+19}{14} = 2 + \tfrac{\sqrt{403}-9}{14}
\end{aligned}$$

Then, just as y_2 gives $x_1 = \frac{\sqrt{403}+20}{3}$, y_4 gives $x_{-1} = \frac{\sqrt{403}+19}{3}$ and y_5 gives $x_{-2} = \frac{\sqrt{403}+9}{14}$.

Combining periodicity with reversibility strengthens Theorem A.1.1 (h).

[3] Since y_0 meets (A.4) and the same recursive formula is applied, it is clear that Theorem A.1.1 still applies to negative indices.

Lemma A.1.5. *There exists a positive integer π such that $\forall i$ $x_i = x_{i+\pi}$, i not necessarily positive.*

Proof: From the proof of Theorem A.1.1 (h) there are k and π such that $\forall i \geq k$, $x_i = x_{i+\pi}$. Essentially, this is equivalent to proving that there is no lower bound for k. Assume the contrary, that there is some lower bound k. Let k and π be the smallest such integers. Then $x_k = x_{k+\pi}$. But by Lemma A.1.4 $x_{k-1} = x_{k+\pi-1}$, so that $k-1$ also meets this criteria, violating the assumption that k is the smallest such integer. Therefore, $\forall i$ $x_i = x_{i+\pi}$. **QED**

Throughout, π will consistently denote the period, even when considering this period in the context of quadratic forms or lattices.

Often the continued fraction may have other characteristics that are interesting besides its periodicity. For factorization, continued fractions with symmetries, such as at x_0 and x_5 from Example 10, will be especially important. If the starting condition near some point is the same in both directions, the entire sequence will be symmetric about that point. This is the point of Lemma A.1.6.

Lemma A.1.6. *Let $x_0 = \frac{\sqrt{N}+P_{-1}}{Q_0}$ meet (A.4) such that $Q_0 \mid 2P_{-1}$. The sequence of pseudo-squares is symmetric about Q_0, so that $\forall i$ $Q_i = Q_{-i}$.*

Proof:

Observe that $0 < \sqrt{N} - P_0 < Q_0$, with $P_0 = b_0Q_0 - P_{-1}$, so that

$$0 < \sqrt{N} - b_0Q_0 + P_{-1} < Q_0.$$

There can only be one possible integer value of b_0 that satisfies this inequality. Since $0 < \sqrt{N} - P_{-1} < Q_0$, $b_0 = 2P_{-1}/Q_0$ satisfies this inequality, so that $P_0 = P_{-1}$.

Let $y_{-1} = \frac{\sqrt{N}+P_1}{Q_1}$. Then, by Lemma 2, $y_0 = \frac{\sqrt{N}+P_0}{Q_0} = \frac{\sqrt{N}+P_{-1}}{Q_0} = x_0$

Therefore, the sequence of pseudo-squares will be symmetric about Q_0, since in either direction the first continued fraction term is the same. Therefore, $Q_i = Q_{-i}$. **QED**

The presence of one point of symmetry allows a proof that another point of symmetry exists and that a factorization of N may be obtained from this symmetry[4]:

Theorem A.1.7. *Let $s = \lfloor \frac{\pi}{2} \rfloor$, where π is the period from Lemma A.1.5. If π is even, $\forall i\ Q_{s+i} = Q_{s-i}$, but $Q_s \neq Q_0$ and $Q_s \mid 2N$. If π is odd, $\forall i\ Q_{s+i+1} = Q_{s-i}$ and either* $\gcd(Q_s, N)$ *is a nontrivial factor of N or -1 is a quadratic residue of N.*

Proof:

Case 1: If π is even, $\pi = 2s$. Then, by Lemmas A.1.6 and A.1.5, $Q_{s+i} = Q_{-s-i} = Q_{2s-s-i} = Q_{s-i}$. Since $Q_{s+1} = \frac{N-P_s^2}{Q_s}$ and $Q_{s-1} = \frac{N-P_{s-1}^2}{Q_s}$, this simplifies to $P_s^2 = P_{s-1}^2$, but since $\forall i\ P_i > 0$, this provides $P_s = P_{s-1}$.

Now $Q_s = \frac{P_s+P_{s-1}}{b_s} = \frac{2P_s}{b_s}$, so that $Q_s \mid 2P_s$.

Assume $Q_s = Q_0$. If Q_s is even, then $P_0 \equiv P_s \equiv 1 \pmod 2$ and if Q_s is odd, $Q_s \mid P_s$. Either way, there is then a unique integer in the range $(\sqrt{N} - Q_0, \sqrt{N})$ satisfying these conditions, so that $P_s = P_0$. Therefore, $x_s = \frac{\sqrt{N}+P_0}{Q_0} = x_0$, contradicting the fact that π is the smallest positive integer such that $\forall i\ Q_i = Q_{i+\pi}$. Therefore, $Q_s \neq Q_0$.

Now $N = P_s^2 + Q_s Q_{s+1}$, so it is apparent that if Q_s is odd, then $Q_s \mid P_s$, so that $Q_s \mid N$. Conversely, if Q_s is even, then $(Q_s/2) \mid P_s$, so that $(Q_s/2) \mid N$. Either way, $Q_s \mid 2N$.

Case 2: If π is odd, $\pi = 2s+1$. Then, by Lemma A.1.6 and A.1.5, $Q_{s+i+1} = Q_{-s-i-1} = Q_{2s+1-s-i-1} = Q_{s-i}$.

Specifically, $Q_s = Q_{s+1}$, so that $N = P_s^2 + Q_s Q_{s+1} = P_s^2 + Q_s^2$, so that $P_{s+1}^2 \equiv -Q_s^2 \pmod N$. If $\gcd(Q_s, N) > 1$, this is a nontrivial factor of N, and the proof is done. Therefore, assume that Q_s and N are relatively prime, so that $Q_s^{-1} \pmod N$ exists. Then $(Q_s^{-1})^2 P_{s+1}^2 \equiv -1 \pmod N$. Then $Q_s^{-1} P_{s+1}$ is a square root of -1 modulo N. **QED**

One final concept that will appear much more important in later sections is equivalence. Define the set $\mathbb{T}$ to be set of all numbers of the form $\frac{\sqrt{N}+P}{Q}$ such that:

$$P^2 \equiv N \pmod Q, \tag{A.7}$$

[4] This was actually discovered in the opposite order. It was clear that ambiguous forms that met this criterion provided a factorization but was later realized that these same forms produced symmetry points. This was first noticed by Gauss [8] and first applied by Shanks [23].

Then define

$$\mathbb{T}^* = \{x \in \mathbb{T} : 0 < P < \sqrt{N},\ |\sqrt{N} - Q| < P\}.$$

An element $x \in \mathbb{T}$ is *reduced* if $x \in \mathbb{T}^*$. For $x, y \in \mathbb{T}^*$, x is *equivalent* to y if x appears in the same continued fraction expansion as y and it is trivial that this is an equivalence relation on $\mathbb{T}^*$. Extending this to all of $\mathbb{T}$ requires a lemma relating elements of $\mathbb{T} - \mathbb{T}^*$ with elements of $\mathbb{T}^*$.

Lemma A.1.8. *Let $x = x_0 \in \mathbb{T} - \mathbb{T}^*$. x_0 may be reduced by applying*

$$x_{i+1} = \frac{1}{x_i - b_i},\ b_i = \lfloor x_i - 1/2 \rfloor,\ i \geq 0 \tag{A.8}$$

until $|Q_i| < 2\sqrt{N}$ for some i and then applying equation (A.2) normally until $x_k \in \mathbb{T}^*$ for some $k > 0$.

Proof: The choice of b_i yields that $\left|\frac{\sqrt{N}-P_i}{Q_i}\right| < \frac{1}{2}$ after the first step, so that $|P_i| < \frac{1}{2}|Q_i| + \sqrt{N}$. Therefore,

$$\begin{aligned} |Q_{i+1}| &= \left|\tfrac{N-P_i^2}{Q_i}\right| \\ &= \left|\tfrac{\sqrt{N}-P_i}{Q_i}\right| |\sqrt{N} + P_i| \\ &< \tfrac{1}{2}(2\sqrt{N} + \tfrac{1}{2}|Q_i| \\ &= \sqrt{N} + \tfrac{1}{4}|Q_i| \end{aligned}$$

so that $|Q_i|$ will decrease as long as $|Q_i| > \frac{4}{3}\sqrt{N}$. When $|Q_r| < 2\sqrt{N}$, revert back to the standard formula for b_r. There are three cases for what Q_r is:

Case 1: $0 < Q_r < \sqrt{N}$. In this case, it is clear that x_{r+1} will be reduced.

Case 2: $\sqrt{N} < Q_r < 2\sqrt{N}$. In this case, if $P_r > 0$, then x_{r+1} will be reduced. Otherwise, $|P_r| < \sqrt{N}$, so that x_{r+1} will be in Case 1.

Case 3: $-2\sqrt{N} < Q_r < 0$. Then $\sqrt{N} < P_r < \sqrt{N} + |Q_r|$, yielding $Q_{r+1} < 2\sqrt{N} + |Q_r|$. If $Q_{r+1} < 2\sqrt{N}$, it is in Case 1 or Case 2. If $2\sqrt{N} < Q_{r+1} < 2\sqrt{N} + |Q_r|$, the choice of b_r provides $\sqrt{N} + P_r > Q_{r+1}$, so that $0 < P_{r+1} < \sqrt{N}$, so that x_{r+2} will be in Case 1. **QED**

I will provide one example of reduction:

Example 12.

$$\begin{aligned} x_0 &= \tfrac{\sqrt{403}+267}{-134} = -2 + \tfrac{\sqrt{403}-1}{-134} \\ x_1 = \tfrac{-134}{\sqrt{403}-1} &= \tfrac{\sqrt{403}+1}{-3} = -8 + \tfrac{\sqrt{403}-23}{-3} \\ x_2 = \tfrac{-3}{\sqrt{403}-23} &= \tfrac{\sqrt{403}+23}{42} = 1 + \tfrac{\sqrt{403}-19}{42} \\ x_3 = \tfrac{42}{\sqrt{403}-19} &= \tfrac{\sqrt{403}+19}{1} = 39 + \tfrac{\sqrt{403}-20}{1} \end{aligned}$$

Lemma A.1.8 defines a map from $\mathbb{T}$ to $\mathbb{T}^*$ (Elements of $\mathbb{T}^*$ are mapped to themselves). Then two elements are equivalent if their corresponding elements of $\mathbb{T}^*$ are equivalent and it is clear that this is still an equivalence relation. Essentially, this equates to saying that two numbers x and y are equivalent if their continued fraction expansions have the same "tail", so that after a certain number of terms of each they have identical cycles. Appendix A.2 will define a different equivalence relation on binary quadratic forms and then prove that it corresponds to this.

A.2 Binary Quadratic Forms

A fuller account of binary quadratic forms can be found in Gauss's [8] and in Buell's [2]. However, here are the necessary fundamental ideas.

A binary quadratic form is a polynomial of the form $F(x,y) = ax^2+bxy+cy^2, x, y, a, b, c \in \mathbb{Z}$ (Often this is abbreviated as (a,b,c)). In some sense then, a quadratic form may be considered to be the set of all the numbers it can represent for various values of x and y. Thus, two quadratic forms are *equivalent* if they represent the same set of integers. It is evident that if one form is transformed into another by the substitution

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix},\ ad - bc = \pm 1, \tag{A.9}$$

then, since this matrix is invertible, the two forms are equivalent. As one further useful distinction, (A.9) is *proper* if its determinant is $+1$ and *improper* if its determinant is -1. The symbol $(\sim)$ will only apply to proper equivalence. For the purpose of factorization, the interesting forms are those that can be improperly transformed into themselves, referred to as *ambiguous forms.*

Example 13. $F(x, y) = -14x^2 + 10xy + 5y^2$ is transformed into itself by the substitution:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -2 & -1 \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix},$$

so $(-14, 10, 5)$ is ambiguous.

Denote the set of all quadratic forms with discriminant Δ by $\mathbb{F}_\Delta$, or often just $\mathbb{F}$. The next obvious question is the organization of all of the quadratic forms equivalent to some given form. Since there are an infinite number of forms equivalent to any form, the search must be narrowed some by first defining reduced forms.

Definition A.2.1. A quadratic form $ax^2+bxy+cy^2$, with positive discriminant $\Delta = b^2-4ac$ is *reduced* if:

$$0 < b < \sqrt{\Delta} \tag{A.10}$$

$$|\sqrt{\Delta} - 2|a|| < b \tag{A.11}$$

Note that $\Delta = b^2 - 4ac$ and (A.10) require that $ac < 0$, so that a and c must have opposite signs.

Making Gauss's description of the organization make some sense will require one more of his definitions:

Definition A.2.2. Two forms $F(x, y) = ax^2 + bxy + cy^2$ and $F'(x, y) = a'x^2 + b'xy + c'y^2$ are *adjacent* if $c = a'$.

To each quadratic form, there is a unique reduced equivalent form adjacent to it on each side[5], and since (A.10-A.11) imply a finite number of possible coefficients, this process

[5] Although Gauss had a recursive mechanism for finding these, continued fractions provide a sufficient mechanism for this that will be defined momentarily. Note that reversal suddenly becomes trivial.

eventually repeats, forming a cycle. The important aspect of this is that the cycle is actually all of the reduced forms equivalent to the first form:

Theorem A.2.3. *[8] If the reduced forms F,F' are properly equivalent, each of them will be contained in the period of the other.*

Gauss proves this in Article 193 of [8], Lenstra proves this in [12], and it is a corollary of Lemma A.4.7 in Appendix A.4. Therefore the proof is omitted.

For now, the important detail is that the quadratic forms correspond directly to the elements of $\mathbb{T}$, and that the reduced quadratic forms correspond to elements of $\mathbb{T}^*$. Note that the elements of $\mathbb{T}$ have attached indices, where the important trait of the index is whether it is odd or even. Define a map from $\mathbb{T}$ to $\mathbb{F}$ by

$$\begin{gathered}\Phi_{\mathbb{T},\mathbb{F}} : \mathbb{T} \to \mathbb{F} \\ \tfrac{\sqrt{N}-P_i}{Q_i} \to F_i(x,y) = Q_i(-1)^i x^2 + 2P_i xy + Q_{i+1}(-1)^{i+1} y^2\end{gathered} \tag{A.12}$$

The inverse map is

$$\begin{gathered}\Phi_{\mathbb{F},\mathbb{T}} : \mathbb{F} \to \mathbb{T} \\ ax^2 + bxy + cy^2 \to x_i = \tfrac{\sqrt{\Delta/4}-b/2}{|a|} \in \mathbb{T}\end{gathered} \tag{A.12$'$}$$

where the discriminant of the quadratic form is Δ is the element of $\mathbb{T}$ is given either an even or odd index as a is positive or negative, respectively. Note that $\Delta = b^2 - 4ac$ gives $4a \mid \Delta - b^2$, so that x_i really is in $\mathbb{T}$.

§A.1 defined an equivalence on $\mathbb{T}$ and suggested that it corresponded with an equivalence of binary quadratic forms. Theorem A.2.4 formalizes this:

Theorem A.2.4. *Under the mapping $\Phi_{\mathbb{T},\mathbb{F}}$, the equivalence classes of $\mathbb{T}$ correspond to the equivalence classes of $\mathbb{F}$. That is, for $x_i, x_j \in \mathbb{T}$ corresponding to $F_i = \Phi_{\mathbb{T},\mathbb{F}}(x_i), F_j = \Phi_{\mathbb{T},\mathbb{F}}(x_j) \in \mathbb{F}$, respectively, $x_i \sim x_j$ if and only if $F_i \sim F_j$.*

Proof:

Let $x_i \sim x_j$. Since x_i and x_j must be in the same continued fraction expansion, assume without loss of generality that $j = i + 1$. The other cases may be easily derived from this case. Then the quadratic form related to x_i is given in (A.12). The substitution

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & (-1)^i b_{i+1} \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix}$$

transforms F_i into

$$Q_{i+1}(-1)^{i+1}x^2 + 2P_{i+1}xy + Q_{i+2}(-1)^{i+2}y^2$$

Observe that this matrix has determinant 1, so that this equivalence is proper.

In order to prove the converse, that the x_i's related to equivalent quadratic forms are equivalent, by Theorem A.2.3, observe that the last coefficient of the quadratic form related to x_i, $Q_{i+1}(-1)^{i+1}$, is then the first coefficient of the quadratic form related to x_{i+1}. Therefore, these two forms are adjacent and thus equivalent. **QED**

The real value of quadratic forms is the composition of quadratic forms. In Article 236 of [8], Gauss provides a very flexible definition of composition. Gauss defines composition as multiplying two quadratic forms together and then making a substitution to simplify this into another binary quadratic form. The algorithm he provides is very complicated, allowing for choices of variables along the way that permit the result to be any quadratic form in the resulting equivalence class. The result of composition should be predictable, so definition needs to be limited some. Shanks and Buell both provide a significant simplification of this algorithm. The symbol $(*)$ will consistently be used for composition.

Proposition 2. [2] Let $F_1 = (a_1, b_1, c_1)$ and $F_2 = (a_2, b_2, c_2)$ be primitive forms of discriminants d_1 and d_2, respectively, such that $d_1 = \Delta n_1^2$ and $d_2 = \Delta n_2^2$ for integers n_1 and n_2 and Δ, with $\Delta = \gcd(d_1, d_2)$. Let

$$m = \gcd(a_1 n_2, a_2 n_1, \frac{b_1 n_2 + b_2 n_1}{2}).$$

Then the congruences

$$mn_1B \equiv mb_1 \pmod{2a_1}$$

$$mn_2B \equiv mb_2 \pmod{2a_2}$$

$$m(b_1n_2 + b_2n_1)B \equiv m(b_1b_2 + \Delta n_1n_2) \pmod{4a_1a_2}$$

are simultaneously solvable[6] for an integer B, and the composition of F_1 and F_2 is:

$$F_1 * F_2 = (\frac{a_1a_2}{m^2}, B, \frac{(B^2 - \Delta)m^2}{4a_1a_2})$$

of discriminant Δ.

See [21] for a derivation of this in the case where the discriminants are equal or [2] for a proof of this case. Buell [2] also provides the substitutions that would be needed for Gauss's definition of composition.

The next question is how this operation is related to equivalence.

Theorem A.2.5. *If $F_1 \sim F_2$, then $F * F_1 \sim F * F_2$.*

Gauss proves this in Article 237-239 of [8].

Therefore, composition treats the equivalence classes in the convenient manner. These equivalence classes are then the elements of the class group, with composition as the group operation. The application of Theorem A.2.5 is that it doesn't matter which form is used to represent an equivalence class.

The significance of ambiguous forms for factorization has been mentioned some above. It is evident that if one form is ambiguous, then its entire equivalence class is also ambiguous. Lemmas A.2.6 generalizes the reasons to be interested in these classes.

Lemma A.2.6. *An ambiguous equivalence class contains two points of symmetry, that is, pairs of reduced adjacent forms, (c, b, a) and (a, b, c) in the cycle that are the symmetric reverse of each other. Let a be the connecting term of either symmetry point. Either a divides the determinant, or $a/2$ divides the determinant.*

Proof:

[6] By convention, choose the answer with the smallest absolute value.

Let A be an ambiguous equivalence class and let $F = ax^2 + bxy + cy^2 \in A$. Let $F' = cx^2 + bxy + ay^2$. Then since $F \in A$, there is a substitution of determinant -1 that maps F into itself. Since the obvious substitution to exchange x and y in F has determinant -1, the product of these two is a proper substitution that transforms F into F'. Therefore, $F' \in A$, so that if F is F_0, F' is F_j for some j. Then that F_1 must be the reverse of F_{j-1}, and so forth. Now, if j is even, then by this process $F_{j/2}$ is its own reverse. However, by the definition of being reduced, the end coefficients of each form must have opposite sign, so this is impossible. Therefore, j must be odd, and then $F_{\frac{j-1}{2}}$ is the reverse of $F_{\frac{j+1}{2}}$.

At this point, observe that since the end-coefficients alternate signs, the entire period must be even. By the same arguments as Theorem A.1.7, one could show that there must be another point of symmetry with the property that $\forall i \; Q_{s+i} = Q_{s-i}$, but such that Q_s is not the same as the connecting term at the first symmetry point. The two quadratic forms containing Q_s as an end coefficient then meet the criteria.

The fact that either a divides the determinant, or $a/2$ divides the determinant was proven in Theorem A.1.7, since the determinant is $4N$. **QED**.

Note that Theorem A.1.7 described two different types of points of symmetry. With the quadratic form cycle, the second case can be ignored because of the alternating signs. However, it is quite possible for the term at one symmetry point to be merely the negative of the term at the other symmetry point. This would correspond to the continued fraction having an odd period and there would be a symmetry point of the second type in the continued fraction at half-way. However, this type of symmetry does not generally provide a factorization for N.

Lastly, it is important how these ambiguous forms fit into the rest of the class group. First, addressing the class group structure requires inverses. Lemma A.2.7 is fairly elementary and is probably stated somewhere else. Let 1 represent the form in the principal cycle whose first coefficient is 1. Let F^{-1} indicate the symmetric reverse of F, $(a, b, c)^{-1} = (c, b, a)$. Lemma A.2.7 justifies this notation:

Lemma A.2.7. $F * F^{-1} \sim 1$

Proof:

Let $F = ax^2 + bxy + cy^2$. Then $F^{-1} = cx^2 + bxy + ay^2$. Let G be the next form adjacent to F^{-1}, that is $G = ax^2 + b'xy + c'y^2$, with $a \mid (b+b')$ from the correspondence with continued fractions. Composing $F * G$, $n_1 = n_2 = 1$ and $m = a$, so that the first coefficient of $F * G$ is 1. Therefore, $F * G \sim 1$, but $F^{-1} \sim G$, so $F * F^{-1} \sim 1$. **QED**.

Note that this implies that the square of a symmetry point is 1.

Theorem A.2.8 was probably known by Shanks, since SQUFOF depends highly on it, but it does not seem that he states this explicitly anywhere.

Theorem A.2.8. *An equivalence class has order* 2 *or* 1 *in the class group if and only if it is ambiguous.*

Proof:

Let A be an ambiguous class. Let $F \in A$. Then $F \sim F^{-1}$, so that $F * F \sim F * F^{-1} \sim 1$. Therefore $F * F$ is in the principal cycle, so that A has order 2 or 1 in the class group.

Conversely, assume that an equivalence class A has order 2 or 1 in the class group. Let $F \in A$. Then $F*F$ is in the principal ideal, so that $F*F \sim (F*F)^{-1}$. But from composition, it is clear that $(F * F)^{-1} \sim F^{-1} * F^{-1}$. So $F * F \sim F^{-1} * F^{-1}$. Since the class group is associative, composing on the right with F maintains equivalence. Therefore:

$$\begin{gathered}(F * F) * F \sim (F^{-1} * F^{-1}) * F \\ 1 * F \sim F^{-1} * (F^{-1} * F) \\ F \sim F^{-1}\end{gathered}$$

Therefore, A is ambiguous. **QED**.

Certainly the class group structure is interesting, but it is now possible to return to the problem from the Morrison-Brillhart algorithm of Example 6 with $Q_3 = 3$, so $Q_6 = 9$ doesn't provide a nontrivial factor of N. The quick explanation is that if you square the quadratic form with first coefficient Q_3, you obtain the quadratic form with first coefficient Q_6. Since the principal cycle is closed under composition, it seems as though, and perhaps would be convenient if, the forms in the principal cycle formed a group. However, the problem of reduction prevents this:

Example 14. Consider the quadratic form $F = (36, 70, -3)$, with determinant $4 \cdot 1333$. Compare $(F * F) * F$, with $F * F * F$, where the difference is that in the first the result is reduced after the first composition. $F * F = (324, -38, -3)$ and the very next adjacent form $(-3, 68, 59)$, is reduced. $F * (-3, 68, 59) = (-12, 70, 9)$, which is already reduced. However, without reduction $F * F * F = F * (324, -38, -3) = (729, 448, -348)$. When this is reduced, the first reduced form found, after 2 steps, is $(9, 56, -61)$.

Therefore, the principal cycle, with the operation being composition followed by reduction, doesn't even meet the requirements for being power associative. However, the observation that the two results are adjacent forms, and that the second reduction took one step longer, prompts us to dig a little deeper.

Understanding this requires what Shanks referred to as infrastructure distance. For $m < n$, and for $x_i \in \mathbb{T}$, the terms in the continued fraction in (A.5), Shanks defined infrastructure distance by

$$D_{\mathbb{F}}(x_m, x_n) = \log\left(\prod_{k=m+1}^{n} x_k\right) \tag{A.13}$$

Lenstra [12] adds a term of $\frac{1}{2}\log(Q_n/Q_m)$ to this, with the effect that the resulting formulas are slightly simplified but the proofs are more complicated and less intuitive. This definition is used by Williams in [26].

Since the quadratic forms are cyclic, in order for the distance between two forms to be measured consistently, it must be considered modulo the distance around the principal cycle.

Definition A.2.9. Let π be the period of the principal cycle. The *regulator* R of the class group is the distance around the principal cycle, that is,

$$R = D_{\mathbb{F}}(F_0, F_\pi) = D(1, F_\pi)$$

Therefore, distance must be considered modulo R, so that $D_{\mathbb{F}}$ is a map from pairs of forms to the interval $[0, R) \in \mathbb{R}$. The addition of two distances must be reduced modulo R

as necessary.

Further analysis of this distance will require two more tools: ideals and lattices. In order to relate to continued fractions, the ideals will be in $\mathbb{Z}[\sqrt{N}] = \{a + b\sqrt{N} :\ a, b \in \mathbb{Z}\}$ and the lattices will be in $\mathbb{Q}(\sqrt{N}) = \{a + b\sqrt{N} :\ a, b \in \mathbb{Q}\}$where N is a non-square positive integer.

A.3 Ideals

Remark: The ideals in $\mathbb{Z}[\sqrt{N}]$ typically correspond only to quadratic forms of discriminant $4N$. Note that if $N \equiv 1 \pmod 4$, then $\mathbb{Z}[\sqrt{N}]$ is not the ring of integers for $\mathbb{Q}(\sqrt{N})$. For $N \equiv 1 \pmod 4$, an analysis of ideals in $\mathbb{Z}[\frac{\sqrt{N}+1}{2}]$ is also interesting, but will be avoided in the interest of simplicity. Quadratic forms of discriminant $N \equiv 1 \pmod 4$ may be related to ideals in $\mathbb{Z}[\sqrt{N}]$ via first multiplying by 2 to obtain quadratic forms of discriminant $4N$.

For $\xi \in \mathbb{Q}(\sqrt{N})$, let $\overline{\xi}$ refer to the *conjugate* of ξ (i.e. $\overline{1 + \sqrt{3}} = 1 - \sqrt{3}$).

The *norm* of a number in $\mathbb{Q}(\sqrt{N})$ is $\mathcal{N}(\xi) = \xi\overline{\xi} \in \mathbb{Q}$.

To simplify notation, the symbols H, I, J, and K will consistently be ideals, u and v will be elements of ideals, α and β will be elements of $\mathbb{Z}[\sqrt{N}]$, ξ and ζ will be elements of $\mathbb{Q}(\sqrt{N})$, and $\mathcal{L}$ will be a lattice.

Our definition of an ideal is the same as in any other commutative ring with identity:

Definition A.3.1. A subset I of a ring R is an *ideal* if for $u, v \in I$, $u \pm v \in I$ and for $\alpha \in R$, $u \cdot \alpha \in I$ (that is, I is closed under addition and multiplication by an element of R). If I is an ideal of $R = \mathbb{Z}[\sqrt{N}]$, define $L(I)$ to be the least positive rational integer in I.

Describing ideals will require the notation for the lattice generated by a set. If

$$\alpha_1, \alpha_2, ..., \alpha_k \in \mathbb{Z}[\sqrt{N}],$$

denote[7] the lattice generated by these as

[7] Observe the difference between the use of [...] here and in §A.1. This expression is completely unrelated to rational approximations.

$$[\alpha_1, \alpha_2, ..., \alpha_k] = \{\sum_{i=1}^{k} n_i \alpha_i : n_i \in \mathbb{Z}\} \tag{A.14}$$

Lemma A.3.2 identifies necessary and sufficient conditions for a set in $\mathbb{Z}[\sqrt{N}]$ to be an ideal.

Lemma A.3.2. *For $Q, s, N, P \in \mathbb{Z}$, N non-square and positive, $[Q, s\sqrt{N} + P]$ is an ideal of the ring $\mathbb{Z}[\sqrt{N}]$ if and only if $sQ \mid \mathcal{N}(s\sqrt{N} + P)$, $s \mid Q$, and $s \mid P$.*

Proof: Assume that $I = [Q, s\sqrt{N} + P]$ is an ideal of $\mathbb{Z}[\sqrt{N}]$. Then, choosing $\alpha = P - s\sqrt{N}$, $\mathcal{N}(s\sqrt{N} + P) \in I$. Since this is an integer, $Q \mid \mathcal{N}(s\sqrt{N} + P)$. Choosing $\alpha = \sqrt{N}$, $Q\sqrt{N} \in I$. Therefore, $s \mid Q$. Since $Q \mid \mathcal{N}(s\sqrt{N} + P)$, this also implies that $s \mid P$. Therefore, α could have been chosen $\alpha = P/s - \sqrt{N}$ so that $Q \mid \mathcal{N}(s\sqrt{N} + P)/s$, so that $sQ \mid \mathcal{N}(s\sqrt{N} + P)$.

Conversely, let $I = [Q, s\sqrt{N} + P]$ and assume that $sQ \mid \mathcal{N}(s\sqrt{N} + P)$, $s \mid Q$, and $s \mid P$. Closure under addition is trivial. To see that I is closed under multiplication by an element of $\mathbb{Z}[\sqrt{N}]$, one need only consider multiplication by 1 and $\sqrt{N}$, since they form a basis for $\mathbb{Z}[\sqrt{N}]$. Multiplication by 1 is trivial. For $\sqrt{N}$,

$$Q\sqrt{N} = \frac{Q}{s}(s\sqrt{N} + P) - \frac{P}{s}Q$$

and Q/s and $-P/s$ are integers. Also,

$$(s\sqrt{N} + P)\sqrt{N} = sN + P\sqrt{N} = \frac{P}{s}(s\sqrt{N} + P) + (\frac{-P^2 + s^2N}{sQ})Q$$

and $\frac{P}{s}$ and $(\frac{-P^2+s^2N}{sQ})$ are integers. **QED**

If $s = 1$, an ideal is *primitive*. Since $s \mid P$ and $s \mid Q$, ideals that are not primitive will often be written $(s)[Q, \sqrt{N} + P]$. Let $\mathbb{I}$ be the set of all primitive ideals.

Represented in the form $I = [Q, \sqrt{N} + P]$, it is clear that $|Q|$ is the smallest positive rational integer in I. Define

$$L(I) = \min\{I \cap \mathbb{Z}^+\} \tag{A.15}$$

At this point, it is possible to define a correspondence between quadratic forms (of discriminant $\Delta \equiv 0 \pmod 4$) and ideals by:

$$\Phi_{\mathbb{F},\mathbb{I}}(F(x,y) = Ax^2 + Bxy + Cy^2) = [A, \sqrt{(\frac{B}{2})^2 - AC} + \frac{B}{2}] \tag{A.16}$$

$$\Phi_{\mathbb{I},\mathbb{F}}([Q, \sqrt{N} + P]) = F(x,y) = Qx^2 + 2Pxy + (\frac{P^2 - N}{Q})y^2 \tag{A.16$'$}$$

and define a *reduced* ideal as an ideal corresponding to a reduced quadratic form. Note that $\Delta = 4N$.

For example, the quadratic form $(15, 2 \cdot 12, -1)$ corresponds to the ideal $[15, \sqrt{159} + 12]$. The one potential problem that immediately becomes apparent is that while $[15, \sqrt{159} + 12]$ and $[-15, \sqrt{159} + 12]$ are the same ideal, $(15, 2 \cdot 12, -1)$ and $(-15, 2 \cdot 12, 1)$ are different quadratic forms. However, it is apparent that the negative sign is merely carried through composition without affecting the computations. Since each of these forms is in the same location within its respective cycle, this difference will not be important to this investigation of composition and distance.

$\Phi_{\mathbb{T},\mathbb{F}}$ and $\Phi_{\mathbb{F},\mathbb{I}}$ may be combined to obtain

$$\Phi_{\mathbb{T},\mathbb{I}}(\frac{Q}{\sqrt{N} - P}) = [Q, \sqrt{N} - P]$$

and $\Phi_{\mathbb{I},\mathbb{T}}$ is defined in the related obvious way.

If $A = [\alpha_i]$ and $B = [\beta_i]$, $i = 1, 2, 3..., d$, then it is clear that $A = B$ if and only if there exists a $d \times d$ matrix M with determinant ± 1 such that:

$$\langle \alpha_i \rangle = M \langle \beta_i \rangle$$

where $\langle \alpha_i \rangle$ and $\langle \beta_i \rangle$ are vectors.

For these purposes, the most important operation with ideals is their multiplication. Multiplication is defined by

$$[\alpha_i] \cdot [\beta_j] = [\alpha_i \beta_j]$$

For example,

$$I = [15, \sqrt{159} + 12] \cdot [10, \sqrt{159} + 13] = [150, 10\sqrt{159} + 120, 15\sqrt{159} + 195, 315 + 25\sqrt{159}]$$

The 4th component is the sum of the 2nd and 3rd, so it is unnecessary for describing the ideal.

Applying the matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}$$

which has determinant 1, subtracts the 2nd component from the third to obtain

$$I = [150, 10\sqrt{159} + 120, 5\sqrt{159} + 75]$$

The matrix, with determinant 1,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

will subtract twice the 3rd component from the 2nd to obtain

$$I = [150, -30, 5\sqrt{159} + 75]$$

Here the 1st component is a multiple of the 2nd and is thus unnecessary. The answer is simplified to obtain

$$I = [30, 5\sqrt{159} + 75] = 5[6, \sqrt{159} + 15]$$

The process of multiplying ideals can be greatly simplified by several well-known formulae[8] [14].

Theorem A.3.3. *Let $I = [Q, \sqrt{N} + P]$ and $J = [Q', \sqrt{N} + P']$ be ideals of $Q\sqrt{N}$. Let $C = \frac{N-P^2}{Q}$, $C' = \frac{N-(P')^2}{Q'}$. If $\gcd(Q, P, C) = \gcd(Q', P', C') = 1$, then $I \cdot J = s[q, \sqrt{N} + p]$, where*

$$s = \gcd(Q, Q', P + P') \tag{A.17}$$

$$h = \gcd(Q, Q', C, C', 2) \tag{A.18}$$

$$q = hQQ'/s^2 \tag{A.19}$$

$$p \equiv P \pmod{Q/s} \tag{A.20}$$

$$p \equiv P' \pmod{Q'/s} \tag{A.21}$$

$$(P + P')p \equiv N + PP' \pmod{QQ'/s} \tag{A.22}$$

Proof[9]:

Consider the product:

$$I \cdot J = [QQ', Q\sqrt{N} + QP', Q'\sqrt{N} + Q'P, N + PP' + (P + P')\sqrt{N}] \tag{A.23}$$

The smallest integer in $I \cdot J$ may be found by considering the smallest integers that may be produced taking these elements pair-wise.

$$L(I \cdot J) = \gcd(QQ', \operatorname{lcm}(Q, Q')(P - P'), \frac{\operatorname{lcm}(Q, P + P')Q'C'}{P + P'}, \frac{\operatorname{lcm}(Q', P + P')QC}{P + P'})$$

[8] In [14], (A.22) is stated as $(P - p)(P' - p) \equiv n + tp + p^2 \pmod{QQ'/s}$, but in this case $t = 0$ and $n = -N$.

[9] Some of the arguments were taken from Buell's proof in [2] concerning composition of quadratic forms.

Let $s = \gcd(Q, Q', P + P')$, $h = \gcd(Q, Q', C, C', 2)$, $w = hQQ'/s$. Let $f \neq 2$ be a prime. Let a, b, c, d, e, k be the largest possible integers such that $f^a \mid Q$, $f^b \mid Q'$, $f^c \mid (P + P')$, $f^d \mid C$, $f^e \mid C'$, $f^k \mid (P - P')$. Then $f^{a+b} \parallel QQ'$, $f^{\max(a,b)+k} \parallel \operatorname{lcm}(Q, Q')(P - P')$, $f^{\max(a,c)+b+e-c} \parallel \frac{\operatorname{lcm}(Q,P+P')Q'C'}{P+P'}$, and $f^{\max(b,c)+a+d-c} \parallel \frac{\operatorname{lcm}(Q',P+P')QC}{P+P'}$).

The following analysis proves that if $f \neq 2$, the maximum exponent of f in $L(I \cdot J)$ is $a + b - \min(a, b, c)$ while if $h = 2$, then the maximum exponent of 2 in $L(I \cdot J)$ is $a + b + 1 - \min(a, b, c)$, while if $h = 1$, then the maximum exponent of f in $L(I \cdot J)$ is $a + b - \min(a, b, c)$. As this is broken in several different cases, an outline of the proof is helpful:

1) $a = 0$ or $b = 0$ or $c = 0$
2) $a \neq 0$,$b \neq 0$, and $c \neq 0$
  2.1) $f \neq 2$
    2.1.1) $a + d \neq b + e$
        $f \mid (P - P')$
        $f \nmid (P - P')$
    2.1.2) $a + d = b + e$
        $f \mid (P - P')$
        $f \nmid (P - P')$
  2.2) $f = 2$
    2.2.1) $a + d \neq b + e$
        $c > 1,\ k > 1$
        $c > 1,\ k \leq 1$
        $c = 1$
    2.2.2) $a + d = b + e$
        $c > 1,\ k > 1$
        $c = 1$
        $k = 1$

Case 1) If $a = 0$, then $\max(a, c) + b + e - c = b + e \geq b$, $\max(b, c) + a + d - c \geq b$ and $a + b = b$, so the maximum exponent for f in $L(I \cdot J)$ is $b = a + b - \min(a, b, c)$. Similarly, if $b = 0$, then the maximum exponent is $a = a + b - min(a, b, c)$.

Assume $c = 0$. $f^{a+d} \parallel QC = N - P^2$ and $f^{b+e} \parallel Q'C' = N - (P')^2$, subtracting, $f^{\min(a+d,b+e)} \mid (P^2 - (P')^2) = (P + P')(P - P')$. Since $c = 0$, then $f^{\min(a+d,b+e)} \mid (P - P')$. Therefore, $f^{\max(a,b)+\min(a+d,b+e)} \mid \operatorname{lcm}(Q, Q')(P - P')$. However, $\max(a, b) + \min(a+d, b+e) \geq$

$\max(a, b) + \min(a, b) = a + b$. Therefore, the maximum exponent for f in $L(I \cdot J)$ is

$$\min(a+b, \max(a,c)+b+e-c, \max(b,c)+a+d-c) = \min(a+b, a+b+e, a+b+d)$$
$$= a+b = a+b-\min(a,b,c)$$

Case 2.1.1) Assume $c \neq 0$, $f \neq 2$, and $a+d \neq b+e$. Then, $f^{\min(a+d,b+e)} \parallel (P^2 - (P')^2) = (P+P')(P-P')$. If $f \mid (P-P')$, then $f \mid 2P$ and $f \mid 2P'$. Since $f \neq 2$, this gives $f \mid P$, $f \mid P'$. Then, $d = e = 0$. Also, $c \leq \min(a, b)$. Then, the maximum exponent for f in $L(I \cdot J)$ is

$$\min(a+b, \max(a,b)+\min(a,b)-c, \max(a,c)+b-c, \max(b,c)+a-c) =$$

$$a+b-c = a+b-min(a,b,c)$$

If $f \nmid (P-P')$ then $c = \min(a+d, b+d)$ and the maximum exponent for f in $L(I \cdot J)$ is

$$\min(a+b, \max(a,b), \max(a,c)+b+e-c, \max(b,c)+a+d-c)$$

If $a = \min(a,b,c)$, then this is $\min(b, c+b+e-c, \max(b,c)+a+d-c) = b = a+b-\min(a,b,c)$. The case is similar if $b = \min(a,b,c)$. If $c = \min(a,b,c)$, then since $c = \min(a+d, b+e)$, this gives $c = \min(a,b)$. Then the maximum exponent is

$$\min(\max(a,b), a+b+e-c, b+a+d-c) = \max(a,b) = a+b-\min(a,b) = a+b-\min(a,b,c)$$

Case 2.1.2) Assume $c \neq 0$, $f \neq 2$, but $a+d = b+e$. As before, if $f \mid (P-P')$, then $d = e = 0$. In this case also $a = b$. Assume $c \leq a$. Note that $f^{a-c} \mid (P-P')$ and $\max(a,b) + \min(a+d, b+e) - c = a+b-c$. Then the maximum exponent is

$$\min(a+b, \max(a,c)+b-c, \max(b,c)+a-c) = a+b-c = a+b-\min(a,b,c).$$

Alternately, assume $c > a$. Then for some k, $f^k \parallel (P - P')$. The maximum exponent is

$$\min(a+b, \max(a,b)+k, b, \max(b,c)+a-c) = b = a+b-a = a+b-\min(a,b,c)$$

Conversely, assume $f \nmid (P - P')$. Then $c \geq a+d = b+e$ and the maximum exponent is

$$\min(a+b, \max(a,b), \max(a,c)+b+e-c, \max(b,c)+a+d-c) = \min(\max(a,b), a+d)$$
$$= \max(a,b) = a+b-\min(a,b) = a+b-\min(a,b,c)$$

Case 2.2.1) Let $f = 2$. Assume $a+d \neq b+e$. Then $2^{\min(c,k)} \parallel 2P$ and $2^{\min(c,k)} \parallel 2P'$, so that $2^{\min(c,k)-1} \parallel P, P'$. If $c > 1$ and $k > 1$, then $d = e = 0$ and as before the largest exponent is $a+b-\min(a,b,c)$. Assume $c > 1$, $k \leq 1$. Then $k = 1$ and $c = \min(a+d, b+e) - 1$. The largest exponent is then

$$\min(a+b, \max(a,b)+1, \max(a,c)+b+e-c, \max(b,c)+a+d-c)$$

If $a \leq \min(b,c)$ this reduces to $b + \min(e,1) = a+b+\min(e,1) - \min(a,b,c)$. $c+1 \leq a+d \leq c+d$, so $d \geq 1$. Note that if $e \geq 1$ then this is a special case and $h = 2$. If $e = 0$, it is the same as before. The cases when $b \leq \min(a,c)$ are similar.

If $c \leq \min(a,b)$, this exponent is $\min(\max(a,b)+1, a+b+e-c, b+a+d-c)$. Without loss of generality, assume $a+d > b+e$ so that $c = a+d-1 \geq c+d-1$, so that $d = 1$ and $a = c$. Then the exponent is $\min(b+1, b+e, b+d) = b+\min(1,e,d) = a+b+\min(1,e,d)-\min(a,b,c)$. Note again that if $e \geq 1$ and $d \geq 1$, then this is the special case where $h = 2$. Otherwise, it is the same as before.

Assume $c = 1$. Then $k \geq 1$. Then the exponent is

$$\min(a+b, \max(a,b)+\min(a+d, b+e)-1, a+b+e-1, a+b+d-1).$$

If $d = 0$ or $e = 0$, $h = 1$ and this is $a + b - 1 = a + b - \min(a, b, c)$. Otherwise, $h = 2$ and the exponent is $a + b = a + b + 1 - \min(a, b, c)$.

Case 2.2.2) Lastly, assume that $c \neq 0$ but $a + d = b + e$. For some k, $2^k \parallel (P - P')$. $c + k \geq a + d$. If $c > 1$ and $k > 1$, then $d = e = 0$, $a = b$. The exponent is then $\min(2a, a + k, \max(a, c) + b - c)$. If $c > a$, this is $\min(2a, a + k, a) = a = a + b - \min(a, b, c)$. If $c \leq a$, the exponent is $\min(2a, a + k, 2a - c) = 2a - c = a + b - \min(a, b, c)$.

Alternately, if $c = 1$, then $k \geq a + d - 1$ and the exponent is

$$\min(a+b, \max(a, b)+k, a+b+e-1, b+a+d-1) = \min(a+b, a+b+e-1, a+b+d-1).$$

If $e > 0$ and $d > 0$, $h = 2$ and this exponent is $a + b = a + b + 1 - \min(a, b, c)$. If $e = 0$ or $d = 0$, $h = 1$ and this is $a + b - 1 = a + b - \min(a, b, c)$.

If $k = 1$ then $c \geq 1$ and specifically $c \geq a + d - 1 = b + e - 1$. If $c \leq \min(a, b)$, then $d = e = 1$ so that $h = 2$, $c = a = b$ and the exponent is

$$\min(2a, a + 1) = a + 1 = a + b + 1 - \min(a, b, c)$$

If $a \leq \min(b, c)$, then $d \geq e$ and the exponent is

$$\min(a + b, b + 1, b + e, \max(b, c) + a + d - c) = \min(b + 1, b + e)$$

If $e \geq 1$, then $d \geq e \geq 1$, so $h = 2$ and in this case the exponent is $b+1 = a+b+1-\min(a, b, c)$. If $e = 0$, $h = 1$ and in this case the exponent is $b = a + b - \min(a, b, c)$.

Therefore,

$$L(I \cdot J) = hQQ'/s$$

so that for $f \neq 2$, the highest exponent is $a + b - \min(a, b, c)$ and for $f = 2$, the highest exponent is $a + b - \min(a, b, c)$ if $h = 1$ and $a + b + 1 - \min(a, b, c)$ if $h = 2$. Note that this is still divisible by s. After that s is factored out, the result is $q = hQQ'/s^2$.

There are integers t, u, and v such that $tQ + uQ' + v(P + P') = s$. First, consider divisibility by s. This is trivial for every term except $N + PP'$. By the definition of s,

$$\begin{gathered} P + P' \equiv 0 \pmod{s} \\ P' \equiv -P \pmod{s} \\ PP' \equiv -P^2 \pmod{s} \\ N + PP' \equiv N - P^2 \pmod{s} \end{gathered}$$

and since $s \mid Q$ and $Q \mid (N - P^2)$, $s \mid N + PP'$.

The linear combination of the last three elements with coefficients t, u, and v respectively is:

$$s\sqrt{N} + tQP' + uQ'P + v(N + PP')$$

so that it is evident that after s is factored out, the remaining ideal is primitive. Since this is the element of $I \cdot J$ with the smallest coefficient of $\sqrt{N}$, clearly $p = t(Q/s)P' + u(Q'/s)P + v(N + PP')/s$, modulo $L(I \cdot J)$. Then,

$$\begin{aligned} p &= t(Q/s)P' + (s - tQ - v(P + P'))P/s + v(N + PP')/s \\ &\equiv P + v(N - P^2)/s \pmod{Q/s} \\ &\equiv P \pmod{Q/s} \end{aligned}$$

since $Q \mid (N - P^2)$. By symmetric arguments, $p \equiv P' \pmod{Q'/s}$.

To prove (A.22), consider:

$$\begin{aligned}(P+P')sp &= (P+P')(tQP'+uQ'P+v(N+PP')\\ &= (P+P')(tQP'+uQ'P)+(P+P')(N+PP')v\\ &= (P+P')(tQP'+uQ'P)+(s-tQ-uQ')(N+PP')\\ &= s(N+PP')+tQ((P')^2-D)+uQ'(P^2-N)\\ &\equiv s(N+PP') \pmod{QQ'}\end{aligned}$$

Therefore, $(P+P')p \equiv N+PP' \pmod{QQ'/s}$. **QED**

Observe that when $h=1$ (A.19) could be restated as

$$L(I\cdot J) = L(I)L(J)/s^2 \tag{A.24}$$

remembering that $L(I)$ is defined as the smallest positive rational integer in I. This equation is proven in [26] and will be useful later.

Also observe that for $h=1$, the equations describing the product of two ideals correspond exactly to the composition of two quadratic forms. Shanks notes this in [21]. Therefore, the equations concerning distance and multiplication of ideals will correspond to distance and composition of quadratic forms.

The case when $h=2$ connects composition of quadratic forms of discriminant $\equiv 1 \pmod 4$ to multiplication of ideals. If F and G are two quadratic forms with discriminant $N \equiv 1 \pmod 4$, then $2F$ and $2G$ have discriminant $4N$ and correspond to ideals I_{2F} and I_{2G} in $\mathbb{Z}[\sqrt{N}]$. Multiplying, $h=2$ and $I_{2F}\cdot I_{2G} = I_{2(F*G)}$. Therefore, although this case will not be considered further, it is readily seen that the distance formulas derived from ideals in $\mathbb{Z}[\sqrt{N}]$ will still correspond to composition of quadratic forms of discriminant $\equiv 1 \pmod 4$.

A.4 Lattices

Consider lattices in $\mathbb{Q}(\sqrt{N}) \times \mathbb{Q}(\sqrt{N})$. Define $\mathbb{L}$ as the set of all lattices in $\mathbb{Q}(\sqrt{N}) \times \mathbb{Q}(\sqrt{N})$. Define the map $M : \mathbb{Q}(\sqrt{N}) \to \mathbb{Q}(\sqrt{N}) \times \mathbb{Q}(\sqrt{N})$ by

$$M(\xi) = \langle \xi, \overline{\xi} \rangle$$

Multiplication in $\mathbb{Q}(\sqrt{N}) \times \mathbb{Q}(\sqrt{N})$ is defined component-wise, that is, $\langle \xi, \xi' \rangle \cdot \langle \zeta, \zeta' \rangle = \langle \xi\zeta, \xi'\zeta' \rangle$, so that it is clear that M is homomorphic and one-to-one.

Distance will relate to a concept called a minimum:

Definition A.4.1. For a vector $v = \langle v_1, v_2, ...v_d \rangle$, the *normed body* of v, $\mathcal{R}(v)$ is the set

$$\mathcal{R}(v) = \{\langle x_1, x_2, ...x_d \rangle : x_i \in \mathbb{R}, |x_i| < |v_i|, i = 1, 2, ...d\}$$

Abusing notation, denote $\mathcal{R}(\xi) = \mathcal{R}(\langle \xi, \overline{\xi} \rangle)$.

A number ξ (or actually the corresponding vector) is a *minimum* of $\mathcal{L}$ if $\mathcal{R}(\xi) \cap \mathcal{L} = \{0\}$, where 0 is the vector $\langle 0, 0 \rangle$.

A lattice $\mathcal{L}$ is *reduced* if $1 \in \mathcal{L}$ and 1 is a minimum.

For this case with $d = 2$, the normed body is a rectangle in $\mathbb{R}^2$. Note that for $\xi \in \mathbb{Q}(\sqrt{N})$ the normed body $\mathcal{R}(\xi)$ has area equal to four times the absolute value of the norm $|\mathcal{N}(\xi)|$.

To avoid unnecessary generality, this investigation will focus specifically on the lattices corresponding to ideals. Specifically, for the primitive ideal $I = [Q, \sqrt{N} + P]$, define the associated lattice containing 1 in $\mathbb{Q}(\sqrt{N})$ as $\mathcal{L}_I = [1, (\sqrt{N} + P)/Q]$.

Conversely, to each lattice containing 1 in $\mathbb{Q}(\sqrt{N})$ there is an associated primitive lattice (which may or may not be an ideal) in $\mathbb{Z}[\sqrt{N}]$. Equation (A.15) defined the function L. In a similar fashion, for a lattice $\mathcal{L}$, define

$$L(\mathcal{L}) = \min\{n \in \mathbb{Z}^+ : n\mathcal{L} \subset \mathbb{Z}[\sqrt{N}]\} \tag{A.25}$$

Then if $L(\mathcal{L})\mathcal{L}$ is an ideal of $\mathbb{Z}[\sqrt{N}]$ it is the primitive ideal associated to a lattice $\mathcal{L}$. Note that if an ideal I is associated to a lattice $\mathcal{L}_I$, then $L(I) = L(\mathcal{L}_I)$. Define

$$\Phi_{\mathbb{I},\mathbb{L}}([Q, \sqrt{N} + P]) = [1, (\sqrt{N} + P)/Q]$$

and

$$\Phi_{\mathbb{L},\mathbb{I}}(\mathcal{L}) = L(\mathcal{L})\mathcal{L}$$

Note that for some lattices $\mathcal{L}$, $\Phi_{\mathbb{L},\mathbb{I}}(\mathcal{L})$ may not actually be an ideal. Lemma A.4.2 provides conditions for it to be an ideal sufficient for this analysis:

Lemma A.4.2. *Let I be a primitive ideal and let $\mathcal{L} = \Phi_{\mathbb{I},\mathbb{L}}(I)$. If $\mathcal{L}'$ is a lattice with basis $\{1, \xi\}$ and for some θ, $\theta\mathcal{L}' = \mathcal{L}$, then $J = \Phi_{\mathbb{L},\mathbb{I}}(\mathcal{L}')$ is a primitive ideal and*

$$(L(I)\theta)J = (L(J))I$$

Proof: Let $I = [Q, \sqrt{N} + P]$. Then $\mathcal{L} = [1, (\sqrt{N} + P)/Q]$. The statement that $\theta\mathcal{L}' = \mathcal{L}$ requires that

$$\theta \begin{bmatrix} 1 \\ \xi \end{bmatrix} = T \begin{bmatrix} 1 \\ (\sqrt{N} + P)/Q \end{bmatrix}$$

where T is a 2×2 matrix with determinant ± 1. Multiplying by $L(I) = L(\mathcal{L}) = Q$ and $L(J) = L(\mathcal{L}')$:

$$Q\theta \begin{bmatrix} L(\mathcal{L}') \\ L(\mathcal{L}')\xi \end{bmatrix} = L(\mathcal{L}')T \begin{bmatrix} Q \\ (\sqrt{N} + P) \end{bmatrix}$$

so that $(L(I)\theta)J = (L(J))I$. Therefore, J is an ideal. It is primitive by the definition of $\Phi_{\mathbb{L},\mathbb{I}}$. **QED**

For an example of minima, consider the lattice $[1, \sqrt{159}-12]$. $\mathcal{R}(1)$ is a square with sides of length 2 centered at the origin and a simple graph demonstrates that 0 is the only point in the lattice and contained in this square. Therefore 1 is a minimum. $\sqrt{159} - 12$ is also a minimum. $\mathcal{R}(\sqrt{159} + 12)$ is a narrower and taller rectangle also centered at the origin.

Given two minima, it is important to be able to determine whether or not there is another minimum between them. In vector format, if $\langle x_1, y_1\rangle$ and $\langle x_2, y_2\rangle$ are minima with $|x_1| > |x_2|$ and $|y_1| < |y_2|$, these two minima are *adjacent* if there does not exist another minima $\langle x_3, y_3\rangle$ such that $|x_2| < |x_3| < |x_1|$ and $|y_1| < |y_3| < |y_2|$.

Voronoi developed a method (and a theorem) concerning adjacent minima ([4], [26]).

Theorem A.4.3. *Let $\mathcal{L}$ be a lattice with $\{\xi, \zeta\}$ as a basis, where $\xi, \zeta \in \mathbb{Q}(\sqrt{N})$ and suppose that $\zeta > \xi > 0$. Then ζ and ξ are adjacent minima of $\mathcal{L}$ if and only if $|\overline{\xi}| > |\overline{\zeta}|$ and $\overline{\zeta\xi} < 0$.*

Proof: Assume ξ and ζ are adjacent minima. Since they are both minima, $|\overline{\xi}| > |\overline{\zeta}|$, or else ζ would not be a minima. Also $0 < \zeta - \xi < \zeta$. Since ζ is a minima, this requires that $|\overline{\zeta} - \overline{\xi}| > |\overline{\zeta}|$. If $\overline{\zeta}$ and $\overline{\xi}$ had the same sign, this would not be possible. Therefore, $\overline{\zeta\xi} < 0$.

Conversely, assume that $|\overline{\xi}| > |\overline{\zeta}|$ and $\overline{\zeta\xi} < 0$. Assume that ξ is not a minimum of $\mathcal{L}$. Then there exists some $\omega \in \mathbb{Q}(\sqrt{N})$ such that $|\omega| < \xi$ and $|\overline{\omega}| < |\overline{\xi}|$. Since $\omega = a\xi + b\zeta$ for some $a, b \in \mathbb{Z}$, $|a\xi + b\zeta| < \xi$ and $|a\overline{\xi} + b\overline{\zeta}| < |\overline{\xi}|$. If $ab = 0$, then either $a = 0$ or $b = 0$. If $a = 0$, then the second statement contradicts the hypothesis. If $b = 0$, then the first statement gives $\xi < \xi$, clearly false. However, if $ab > 0$ then $|a\xi + b\zeta| > \xi$ and if $ab < 0$, then since $\overline{\zeta\xi} < 0$, $|a\overline{\xi} + b\overline{\zeta}| > |\overline{\xi}|$. Therefore, ξ must be a minima. By similar reasoning, ζ must be a minima.

Concerning adjacency, assume that there is another minima ω between ξ and ζ. Since $\omega = a\xi + b\zeta$ for some $a, b \in \mathbb{Z}$, $\xi < |a\xi + b\zeta| < \zeta$ and $|\overline{\zeta}| < |a\overline{\xi} + b\overline{\zeta}| < |\overline{\xi}|$. Since $\zeta > \xi > 0$, the first statement requires that $b = 0$ and then the second statement simplifies to $|a| < 1$, requiring that $a = 0$ and providing a contradiction. Therefore, ξ and ζ are adjacent minima. **QED**

From the previous example, it is now possible to check that $\xi = 1$ and $\zeta = \sqrt{159} + 12$ are indeed adjacent minima.

The idea that will actually connect to continued fractions (and distance) is the search for a sequence of adjacent minima. This sequence is formed by relating different lattices. The following Lemmas are due to Williams [26].

Lemma A.4.4. *Let $\mathcal{L}$ and $\mathcal{L}'$ be reduced lattices. If $\xi\mathcal{L}' = \mathcal{L}$, then ξ is a minimum of $\mathcal{L}$.*

Proof: Since $1 \in \mathcal{L}'$, $\xi \in \mathcal{L}$. If ξ is not a minimum of $\mathcal{L}$, then there exists a $\zeta \in \mathcal{L}$ such that $\zeta \neq 0$ and $|\zeta| < |\xi|$ and $|\overline{\zeta}| < |\overline{\xi}|$. Let $\beta = \zeta/\xi$, so that $\beta \in \mathcal{L}'$. $|\beta| = |\zeta/\xi| < 1$ and $|\overline{\beta}| = |\overline{\zeta}/\overline{\xi}| < 1$, contradicting the fact that $\mathcal{L}'$ is reduced. Therefore, ξ is a minimum of $\mathcal{L}$. **QED**

Now consider the converse of this statement. Note that $\lfloor x \rfloor$ denotes the floor of x.

Lemma A.4.5. *Let $\mathcal{L} = [1, \xi]$, where 1 and ξ are adjacent minima of $\mathcal{L}$ with $1 > \xi > 0$. Let $\mathcal{L}' = (1/\xi)\mathcal{L}$. Then $\mathcal{L}'$ is a reduced lattice.*

Proof: $\mathcal{L}' = (1/\xi)[1, \xi] = [1/\xi, 1] = [1/\xi - \lfloor 1/\xi \rfloor, 1]$, so that $1 \in \mathcal{L}'$. It is sufficient to show that 1 and $\xi' = 1/\xi - \lfloor 1/\xi \rfloor$ are adjacent minima. First, 1 and ξ' are a basis for $\mathcal{L}'$ and $1 > \xi' > 0$. Since $0 < \xi < 1$, $\lfloor 1/\xi \rfloor > 1$. Since $\overline{\xi} < 0$, $\overline{\xi'} = 1/\overline{\xi} - \lfloor 1/\xi \rfloor < 0 - 1 = -1$. Thereby satisfying both the requirement that $\overline{\xi}' \cdot 1 < 0$ and the requirement that $|\overline{\xi}'| > 1$. Therefore, by Theorem A.4.3, 1 and ξ' are adjacent minima of $\mathcal{L}'$ and thus $\mathcal{L}'$ is a reduced lattice. **QED**

Actually, these proofs provide a bit more by actually finding the minimum adjacent to 1 in the new lattice. The next Lemma makes use of this minimum [26]:

Lemma A.4.6. *Let $\mathcal{L}$, $\mathcal{L}'$, ξ, and ξ' be as above. Let ζ be the minimum adjacent to ξ other than 1 in $\mathcal{L}$. Then $\zeta = \xi\xi'$.*

Proof: $\xi\xi' = \xi(1/\xi - \lfloor 1/\xi \rfloor) = 1 - \xi\lfloor 1/\xi \rfloor$, so that $[\xi, \xi\xi']$ is a basis for $\mathcal{L}$. Since $1 > \xi' > 0$, $\xi > \xi\xi' > 0$. Since $|\overline{\xi'}| > 1$, $|\overline{\xi\xi'}| > |\overline{\xi}|$. Since $\overline{\xi'} < 0$, $\overline{\xi \cdot \xi\xi'} = \overline{(\xi)^2\xi'} < 0$. Therefore, by Theorem A.4.3, ξ and $\xi\xi'$ are adjacent minima. Since $\xi\xi' \neq 1$, $\zeta = \xi\xi'$. **QED**

Observe that by a similar process, one could find a reduced lattice $\mathcal{L}'' = 1/\xi'\mathcal{L}'$, etc. Then $\mathcal{L}'' = 1/(\xi\xi')\mathcal{L}$. To generalize, define $\xi = \xi_1$ and $\mathcal{L} = \mathcal{L}_1$ and this is a sequence of reduced lattices and their minima. A chain of adjacent minima of $\mathcal{L}_1$ may be defined by

$$\theta_n = \prod_{i=1}^{n-1} \xi_i \tag{A.26}$$

and then

$$\theta_n \mathcal{L}_n = \mathcal{L}_1 \tag{A.27}$$

Since each $\mathcal{L}_n$ is a reduced lattice, by Lemma A.4.4 each θ_n is a minimum of $\mathcal{L}_1$.

Although it is not true in higher dimensions, it is fairly trivial in 2-d that this chain of adjacent minima provides a complete (although infinite) list of the minima with x-coordinate between 0 and 1.

Lemma A.4.7. *Let $\langle \phi, \overline{\phi} \rangle$ be a minimum of a lattice $\mathcal{L}$, with $0 < \phi < 1$. Then for some n, $\phi = \theta_n$, where θ_n is defined by equation (A.26)*

Define *distance* in terms of this chain of minima by

$$D_{\mathbb{L}}(\mathcal{L}_n, \mathcal{L}_m) = \log(\theta_n/\theta_m) \tag{A.28}$$

It will become readily apparent that the subscript $\mathbb{L}$ is unnecessary, but it provides clarity for now. Before continuing it is appropriate to provide an example of these concepts. First, as a reference, consider the steps for the continued fraction expansion of $\sqrt{159} - 12$ and the quadratic form distances $D_{\mathbb{F}}$ covered to the end of each step:

$$\begin{aligned}
x_1 &= \tfrac{1}{\sqrt{159}-12} = \tfrac{\sqrt{159}+12}{15} = 1+\tfrac{\sqrt{159}-3}{15}, \quad & D_{\mathbb{F}}(F_0,F_1) &= \log(\tfrac{\sqrt{159}+12}{15})\\
x_2 &= \tfrac{15}{\sqrt{159}-3} = \tfrac{\sqrt{159}+3}{10} = 1+\tfrac{\sqrt{159}-7}{10}, \quad & D_{\mathbb{F}}(F_0,F_2) &= \log(\tfrac{\sqrt{159}+13}{10})\\
x_3 &= \tfrac{10}{\sqrt{159}-7} = \tfrac{\sqrt{159}+7}{11} = 1+\tfrac{\sqrt{159}-4}{11}, \quad & D_{\mathbb{F}}(F_0,F_3) &= \log(\tfrac{2\sqrt{159}+25}{11})\\
x_4 &= \tfrac{11}{\sqrt{159}-4} = \tfrac{\sqrt{159}+4}{13} = 1+\tfrac{\sqrt{159}-9}{13}, \quad & D_{\mathbb{F}}(F_0,F_4) &= \log(\tfrac{3\sqrt{159}+38}{13})\\
x_5 &= \tfrac{13}{\sqrt{159}-9} = \tfrac{\sqrt{159}+9}{6} = 3+\tfrac{\sqrt{159}-9}{6}, \quad & D_{\mathbb{F}}(F_0,F_5) &= \log(\tfrac{5\sqrt{159}+63}{6}).
\end{aligned}$$

The continued fraction corresponds to quadratic forms which correspond to ideals, which are associated with lattices that contain 1. In this case, the lattice associated with x_1 is $\mathcal{L}_1 = [1, 1/x_1] = [1, \sqrt{159} - 12]$ and $1/x_1$ is a minimum adjacent to 1 in $\mathcal{L}_1$. From here:

$$\begin{gathered}
\mathcal{L}_2 = \tfrac{1}{\sqrt{159}-12}\mathcal{L}_1 = [\tfrac{1}{\sqrt{159}-12}, 1] = [\tfrac{\sqrt{159}+12}{15}, 1] = [1, \tfrac{\sqrt{159}-3}{15}] = [1, 1/x_2]\\
\mathcal{L}_3 = \tfrac{15}{\sqrt{159}-3}\mathcal{L}_2 = [\tfrac{15}{\sqrt{159}-3}, 1] = [\tfrac{\sqrt{159}+3}{10}, 1] = [1, \tfrac{\sqrt{159}-7}{10}] = [1, 1/x_3]\\
\cdots
\end{gathered}$$

and it is apparent that this same pattern of correspondence will continue, that is

$$\Phi_{\mathbb{T},\mathbb{L}}(x_n) = \Phi_{\mathbb{T},\mathbb{I}}(\Phi_{\mathbb{I},\mathbb{L}}(x_n)) = [1, 1/x_n] = \mathcal{L}_n \tag{A.29}$$

from which it is also apparent that the sequences of lattices will be periodic.

Computing equation (A.26), for example, $\theta_3 = (\sqrt{159} - 12)(\frac{\sqrt{159}-3}{15}) = 13 - \sqrt{159}$. With $\theta_1 = 1$,

$$D(\mathcal{L}_1, \mathcal{L}_3) = \log(1/(13 - \sqrt{159})) = \log((\sqrt{159} + 13)/10)$$

It is readily apparent that the definition of distances in lattices corresponds to the definition given for quadratic forms. Note that these distances must still be considered modulo R, the regulator, since the sequence of lattices is still cyclic.

A.5 The Generalized Distance Formula

Going back to ideals, note that if $I_1 = L(\mathcal{L}_1)\mathcal{L}_1$ and $I_n = L(\mathcal{L}_n)\mathcal{L}_n$ is another ideal corresponding to a lattice later in the same sequence, then

$$\theta_n \mathcal{L}_n = \mathcal{L}_1$$

$$L(\mathcal{L}_1)L(\mathcal{L}_n)\theta_n \mathcal{L}_n = L(\mathcal{L}_1)L(\mathcal{L}_n)\mathcal{L}_1$$

$$(L(\mathcal{L}_1)\theta_n)I_n = (L(\mathcal{L}_n)I_1 \tag{A.30}$$

where once again, the distance (this time between ideals) is given by $D(I_1, I_n) = -\log(\theta_n)$. Now, this definition of distance is well and good for reduced ideals, but as of yet, it hasn't been applied it to non-reduced ideals. To relate the definitions of reduced lattices and continued fractions observe that the definition of a reduced continued fraction implies that for a term $x_i = \frac{\sqrt{N}+P_{i-1}}{Q_i}$, being reduced equates to

$$\frac{\sqrt{N} + P_{i-1}}{Q_i} > 1$$

$$0 < \frac{\sqrt{N} - P_{i-1}}{Q_i} < 1$$

so that it is clear that if $\mathcal{L}_x = [1, 1/x]$, then 1 and x are adjacent minima and the lattice is reduced. The process of dealing with a non-reduced lattice correlates to the process of reducing a continued fraction as demonstrated in the proof of Lemma A.1.8. See [26] for a more general Lemma.

Lemma A.5.1. *Let I be any primitive ideal in $\mathbb{Z}[\sqrt{N}]$. There exists a reduced ideal I_n and a $\theta_n \in I$ such that*

$$(L(I)\theta_n)I_n = (L(I_n))I \tag{A.31}$$

Proof:

Let $I = [Q, \sqrt{N} + P]$. Then the associated lattice is $\mathcal{L}_I = [1, \frac{\sqrt{N}+P}{Q}] = [1, \xi_1]$. If I is reduced, $I_n = I$, $u = L(I)$, and the proof is done. If I is not reduced, then $\mathcal{L}_I$ is not reduced. Without loss of generality, assume that $0 < \xi_1 < 1$ (since otherwise it would just have to be reduced by an integer.). Let $\mathcal{L}_2 = 1/\xi_1 \mathcal{L}_I = [1/\xi, 1] = [1, 1/\xi - \lfloor 1/\xi - 1/2 \rfloor]$. Then $\xi_1 \mathcal{L}_2 = \mathcal{L}_I$. Continuing in similar manner[10], by Lemma A.1.8 and the correspondence between lattices and continued fractions for some n,ξ_n reduced, and thus $\mathcal{L}_n$ reduced. As in (A.26), set

$$\theta_n = \prod_{i=1}^{n-1} \xi_i$$

so that

$$\theta_n \mathcal{L}_n = \mathcal{L}_I$$

Then $(L(I)\theta_n)I_n = (L(I_n)I$. **QED**

Let I_1, J_1 be reduced primitive ideals. Let K_1 be the primitive ideal found by multiplying I_1 and J_1 and removing a factor and let s be the factor removed, so that $(s)K_1 = I_1 J_1$, $s \in \mathbb{Z}$. By Lemma A.5.1 there exists a reduced ideal K_j and a $\lambda_j \in K_1$ such that

$$(L(K_1)\lambda_j)K_j = (L(K_j))K_1 \tag{A.32}$$

corresponding to $D(K_1, K_j) = -\log(\lambda_j)$.

Let $I_n \sim I_1$ and $J_m \sim J_1$ and let H_1 be the primitive ideal found by multiplying I_n and

[10] Note that it is irrelevant whether or not the second components of the intermediate lattices are either minima or adjacent to 1. Also note that, as in Lemma A.1.8, the formula would change slightly when the denominators get small.

J_m and removing a factor and let t be the factor removed, so that $(t)H_1 = I_nJ_m$, $t \in \mathbb{Z}$. By Lemma A.5.1 there exists a reduced ideal H_k and a $\eta_k \in K_1$ such that

$$(L(H_1)\eta_k)H_k = (L(H_k))H_1 \tag{A.33}$$

corresponding to $D(H_1, H_k) = -\log(\eta_k)$.

Also, there exist minima μ_n and ϕ_m in the lattices corresponding to I_1 and J_1, respectively, such that

$$(L(I_1)\mu_n)I_n = (L(I_n))I_1 \tag{A.34}$$

and

$$(L(J_1)\phi_m)J_m = (L(J_m))J_1 \tag{A.35}$$

corresponding to $D(I_1, I_n) = -\log(\mu_n)$ and $D(J_1, J_m) = -\log(\phi_m)$.

By combining (A.24) and (A.32)-(A.35):

$$\begin{gathered}
(L(H_k))K_j = \left(\tfrac{L(H_k)L(K_j)}{L(K_1)\lambda_j}\right) K_1 \\
= \left(\tfrac{L(H_k)L(K_j)}{L(K_1)\lambda_j s}\right) I_1J_1 \\
= \left(\tfrac{L(H_k)L(K_j)L(I_1)L(J_1)\mu_n\phi_m}{L(K_1)\lambda_j sL(I_n)L(J_m)}\right) I_nJ_m \\
= \left(\tfrac{L(H_k)L(K_j)s\mu_n\phi_m}{\lambda_j L(I_n)L(J_m)}\right) I_nJ_m \\
= \left(\tfrac{L(H_k)L(K_j)s\mu_n\phi_m t}{\lambda_j L(I_n)L(J_m)}\right) H_1 \\
= \left(\tfrac{L(H_k)L(K_j)s\mu_n\phi_m}{\lambda_j tL(H_1)}\right) H_1 \\
= \left(\tfrac{L(K_j)s\mu_n\phi_m\eta_k}{\lambda_j t}\right) H_k
\end{gathered}$$

Set

$$\psi = \frac{s\mu_n\phi_m\eta_k}{t\lambda_j}$$

and then

$$(L(K_j)\psi)H_k = (L(H_k))K_j \tag{A.36}$$

Since K_j and H_k are reduced, by Lemma A.4.4 ψ is a minimum of the lattice $\mathcal{L}_{K_j}$, so that for some n, $\psi = \theta_n$. Therefore,

$$D(K_j, H_k) = -\log(\psi) = -\log(\mu_n) - \log(\phi_m) - \log(\eta_k) + \log(\lambda_j) - log(s/t)$$

$$= D(I_1, I_n) + D(J_1, J_m) + \zeta$$

where $\zeta = D(H_1, H_j) - D(K_1, K_j) + log(t/s)$ will be small compared to $D(K_j, H_k)$ for m, n large.

By the correspondence between multiplication of ideals and composition of quadratic forms, this result may be restated in terms of forms:

Theorem A.5.2. *If $F_1 \sim F_n$ are equivalent forms and $G_1 \sim G_m$ are equivalent forms and $D_{\rho,1}$ is the reduction distance for $F_1 * G_1$ and $D_{\rho,2}$ is the reduction distance for $F_n * G_m$ and s and t are the factors cancelled in each respective composition, then*

$$D(F_1 * G_1, F_n * G_m) = D(F_1, F_n) + D(G_1, G_n) + \zeta$$

where $\zeta = D_{\rho,2} - D_{\rho,1} + \log(t/s)$.

Example 6 from the Morrison-Brillhart algorithm is in the principal cycle. By Theorem A.5.2 when a form F is composed with itself, the distance from 1 to F is roughly doubled, $d(1, F^2) = 2d(1, F) + \zeta$. Therefore, the index is roughly doubled, since distance is roughly proportional to the difference in indices, so that $F_3 * F_3 = F_6$, and $Q_6 = 9$ is the square of $Q_3 = 3$.

Since the square of any symmetry point has first coefficient 1, observe that if the distance around some cycle were unrelated to the distance around the principal cycle, then this result would be affected by which symmetry point this distance was referenced from. From Definition A.2.9 $R = D(F_0, F_\pi)$ in the principal cycle. At this point, it is clear that the distance in other cycles must be the same.

Lemma A.5.3. *Let A be a primitive ambiguous cycle with a period π. Then,*

$$R = D(F_0, F_\pi)$$

Proof: Let $\{F_i\}$ have period π and let F_0 and $F_{\pi/2}$ be the two symmetry points of A. Then $F_0 * F_0 = 1 = F_{\pi/2} * F_{\pi/2}$, with $D_{\rho,1} = D_{\rho,2} = 0$, s and t the respective first coefficients. Therefore,

$$0 = D(F_0 * F_0, F_{\pi/2} * F_{\pi/2}) = 2D(F_0, F_{\pi/2}) + \log(t/s) = D(F_0, F_\pi)$$

where the 3rd step is obtained from the 2nd by the fact that the product in $D(F_0, F_{\pi/2})$ includes the last denominator t and not the first denominator s.

Therefore, $D(F_0, F_\pi) = nR$. Considering composition of F_0 with forms in the principal cycle, clearly $D(F_0, F_\pi) \leq R$, so that $D(F_0, F_\pi) = R$. **QED**

A.6 Proof of Square Forms Factorization (SQUFOF)

It's not certain how much Shanks may have rigorously proven concerning distances, but based on the understanding he had of distance and infrastructure, he was able to develop Square Forms Factorization. A short example will demonstrate and explain the algorithm: let $N = 3193$. Expanding the continued fraction (principal cycle), $Q_{10} = 49$. The quadratic form for this is $F = 49x^2 + 58xy - 48y^2$. Since 49 is a perfect square, $7x^2 + 58xy - 336y^2$, which reduces with $D_\rho = 0$ to $G = 7x^2 + 100xy - 99y^2$ is a quadratic form whose square is

F. Therefore, by Theorem A.2.8, G is in a class of order 2 or 1, so that G is an ambiguous form, so that there are two points of symmetry in its cycle. Since by Theorem A.5.2, $2D(G_s, G) = D(1, F) \pmod{R}$. So $D(G_s, G) = D(1, F)/2 \pmod{R/2}$. Since the two points of symmetry are $R/2$ away from each other, this means that there is a symmetry point at distance $D(1, F)/2$ behind G. Therefore, a point of symmetry may be found by reversing G and traveling this short distance. Now if the coefficient at this symmetry point is ±1, then there would have been a 7 somewhere before F in the continued fraction expansion. If the coefficient is 2, then this symmetry point could be composed with G to find 14 at an earlier point in the principle cycle. Therefore, the symmetry point provides a nontrivial factor for N. In this case, after 6 steps it provides 31 as a factor of 3193.

The second phase of this algorithm can be made significantly (at least for larger numbers) faster if the quadratic forms from the continued fraction expansion with indices that are powers of 2 are saved. In this example, $F = F_{10}$, so that G is about the 5^{th} form in its cycle[11]. The composition of G^{-1} with F_4 and F_1 is close and a simultaneous search in both direction from there quickly finds the symmetry point. In this case, it is only necessary to store $\log_2 k$ forms for k steps, so that it is more efficient to check each square to see if it works than to check each square root against the previous pseudo-squares to predict whether it will work.

Formally, here is the algorithm for factoring N:

[11] Roughly, since in this case $5 \approx 6$.

```
$Q_0 \leftarrow 1, P_0 \leftarrow \lfloor\sqrt{N}\rfloor, Q_1 \leftarrow N - P_0^2$
$r \leftarrow \lfloor\sqrt{N}\rfloor$
while $Q_i \neq$ perfect square for some $i$ even
    $b_i \leftarrow \left\lfloor \frac{r+P_{i-1}}{Q_i} \right\rfloor$
    $P_i \leftarrow b_i Q_i - P_{i-1}$
    $Q_{i+1} \leftarrow Q_{i-1} + b_i(P_{i-1} - P_i)$
    if $i = 2^n$ for some $n$
        Store $(Q_i, 2 \cdot P_i)$ $F_0 = (\sqrt{Q_i}, 2 \cdot P_{i-1}, \frac{P_{i-1}^2 - N}{Q_i})$
Compose $F_0$ with stored forms according to the
binary representation of $i/2$ and store result to $F_0$.
$F_0 = (A, B, C)$
$Q_0 \leftarrow |A|, P_0 \leftarrow B/2, Q_1 \leftarrow |C|$
$q_0 \leftarrow Q_1, p_0 \leftarrow P_0, q_1 \leftarrow Q_0$
while $P_i \neq P_{i-1}$ and $p_i \neq p_{i-1}$
    Apply same recursive formulas to $(Q_0, P_0, Q_1)$ and $(q_0, p_0, q_1)$
If $P_i = P_{i-1}$, either $Q_i$ or $Q_i/2$ is a nontrivial factor of $N$.
If $p_i = p_{i-1}$, either $q_i$ or $q_i/2$ is a nontrivial factor of $N$.
```

Finding a perfect square that provides a factorization is the slowest part of the algorithm, so the number of steps required to obtain this is a good measure of the total runtime. Let W be the number of forms examined before a square for is found that provides a factorization. In [23], Shanks states that for N having k distinct prime factors[12],

$$E(W) = \ln(8)\frac{2+\sqrt{2}}{2}\frac{\sqrt[4]{N}}{2^k - 2}.$$

[12] Shanks actually let N have $k+1$ distinct prime factors, so the distance looks slightly different but is equivalent.

Shanks did not provide a proof, and it actually wasn't quite right for all N, but in 2004, Jason Gower analyzed the runtime and concluded:

Conjecture 2. [9] Let N be square-free and have k distinct odd prime factors. Let W be the number of forms that SQUFOF must examine before finding a square form that provides a factorization. Then,

$$E(W) \sim \begin{cases} \frac{2(\sqrt{2}+1)\sqrt[4]{N}\log 2}{2^k-2} & if N \equiv 1 \pmod{4}, \\ \frac{3(\sqrt{2}+2)\sqrt[4]{N}\log 2}{2(2^k-2)} & if N \equiv 2 \text{ } or \text{ } 3 \pmod{4} \end{cases} \tag{A.37}$$

Note that for $N \equiv 2$ *or* $3 \pmod{4}$, this is equivalent to Shanks estimate.

Therefore, Square Forms Factorization probably has an average runtime of $O(\sqrt[4]{N})$.

Appendix B

Source Code

B.1 Number Theory Functions

This first program, NumberTheory.c, has all of the basic functions needed for composition of quadratic forms and cycling.

```
#include <stdio.h>
#include <stdlib.h>
#include "gmp.h"

struct form {
        mpz_t Q0;
        mpz_t P;
        mpz_t Q1;
        mpz_t b;
};

//prints a the coefficients of a quadratic form on a single line
void outputF(struct form A)
{
    printf(" ");
    mpz_out_str(stdout,10,A.Q0);
    printf(" ");
    mpz_out_str(stdout,10,A.P);
    printf(" ");
    mpz_out_str(stdout,10,A.Q1);
    printf(" \n");
}

//sets dest = src
void ftof(struct form *dest,struct form src)
{
    mpz_set((*dest).Q0,src.Q0);
    mpz_set((*dest).P,src.P);
    mpz_set((*dest).Q1,src.Q1);
    mpz_set((*dest).b,src.b);
}

//Calculates modular inverses.  Also handles if A and base are not relatively prime.
//Sets d = gcd(A,base).  Sets inv = inverse of A modulo (base/d).
void inverse(mpz_t inv, mpz_t d, mpz_t A, mpz_t base)
{
        mpz_t t,one;
        double one_d = 1;
        mpz_init(t);
        mpz_init_set_d(one,one_d);
        mpz_gcdext(d,inv,t,A,base);
        if(mpz_cmp(d,one))
        {
                mpz_cdiv_q(A,A,d);
                mpz_gcdext(t,inv,t,A,base);
        }
        mpz_clear(t);
        mpz_clear(one);
}

//Solves a system of contruence equations: Chinese remainder theorem.
// Sets soln congruent to EQ[0] modulo EQ[1], EQ[2] modulo EQ[3], etc.
int congruent(mpz_t soln, int size, mpz_t EQ[])
{
        mpz_t modu,cong,d,inv,temp,temp2,prodSoFar;
        int i;
        mpz_init(prodSoFar);
        mpz_init(modu);
        mpz_init(cong);
        mpz_init(d);
        mpz_init(inv);
        mpz_init(temp);
        mpz_init(temp2);


        mpz_set(soln,EQ[0]);
        mpz_set(prodSoFar,EQ[1]);
        for(i = 1; i<size; i++)
        {
                mpz_set(cong,EQ[2*i]);
                mpz_abs(modu,EQ[2*i+1]);
                inverse(inv,d,prodSoFar,modu);
                mpz_sub(temp,soln,cong);
                mpz_cdiv_r(temp2,temp,d);
                if(mpz_sgn(temp2))
                {
                        mpz_clear(prodSoFar);
                        mpz_clear(modu);
                        mpz_clear(cong);
                        mpz_clear(d);
                        mpz_clear(inv);
                        mpz_clear(temp);
                        mpz_clear(temp2);

                        return 0;
                }
                mpz_mul(temp,temp,inv);
                mpz_addmul(soln,temp,prodSoFar);
                mpz_cdiv_q(temp2,modu,d);
                mpz_mul(prodSoFar,prodSoFar,temp2);
                mpz_cdiv_r(soln,soln,prodSoFar);
        }

        mpz_clear(prodSoFar);
        mpz_clear(modu);
        mpz_clear(cong);
        mpz_clear(d);
        mpz_clear(inv);
        mpz_clear(temp);
```

```
        mpz_clear(temp2);

        return 1;
}

//Takes one step.  Note that Qform := [Q1,P1,Q2,b1], all positive.
void cFracStep(struct form *A,mpz_t root)
{
        mpz_t temp;

        mpz_init_set(temp,(*A).P);
        mpz_add((*A).b,root,(*A).P);
        mpz_fdiv_q((*A).b,(*A).b,(*A).Q1);
        mpz_submul((*A).P,(*A).b,(*A).Q1);
        mpz_neg((*A).P,(*A).P);
        mpz_sub(temp,temp,(*A).P);
        mpz_addmul((*A).Q0,temp,(*A).b);
        mpz_swap((*A).Q0,(*A).Q1);

        mpz_clear(temp);
}

//Tests if form A is reduced.
int goodForm(struct form A, mpz_t root)
{
    mpz_t lRoot;
    mpz_init_set(lRoot,root);
        mpz_sub(lRoot,root,A.Q1);
        mpz_abs(lRoot,lRoot);
        if((mpz_cmp(A.P,root)>0)||(mpz_cmp(A.P,lRoot)<0))
                return 0;
        return 1;
}

//Given that r is the sqrt(square) mod base, squares base and finds sqrt(square) mod new base congruent to r
int CompSquare(mpz_t r, mpz_t square, mpz_t base)
{
        mpz_t bs,inv,d,temp1,temp2;

        mpz_init_set(bs,base);
```

```
mpz_mul(base,base,base);
if(!mpz_cmp(base,bs))
{
        mpz_clear(bs);
        return 1;
}
mpz_clear(bs);

mpz_init(inv);
mpz_init(d);
mpz_init(temp1);
mpz_init(temp2);


inverse(inv,d,r,base);
mpz_fdiv_qr(temp1,temp2,square,d);
if(mpz_sgn(temp2))
{
        mpz_clear(inv);
        mpz_clear(d);
        mpz_clear(temp2);
        mpz_clear(temp1);
        return 0;
}


mpz_addmul(r,inv,temp1);

if(mpz_tstbit(r,0))
        mpz_sub(r,r,base);

mpz_cdiv_q_ui(r,r,2);
mpz_fdiv_r(r,r,base);



mpz_clear(temp1);
mpz_clear(temp2);
mpz_clear(inv);
mpz_clear(d);
```

```
        return 1;
}

//Reverses a quadratic form and takes a step so that the first coefficient is unchanged.
void reverse(struct form *A,mpz_t root)
{
        mpz_swap((*A).Q0,(*A).Q1);
        cFracStep(A,root);
}

//Reduces A.  Does not use the original value of A.Q1
void reduce(struct form *A, mpz_t root, mpz_t N)
{
        mpz_t temp;
        mpz_init(temp);

        mpz_sub((*A).b,root,(*A).P);
        mpz_fdiv_q((*A).b,(*A).b,(*A).Q0);
        mpz_addmul((*A).P,(*A).b,(*A).Q0);
        mpz_mul(temp,(*A).P,(*A).P);
        mpz_sub((*A).Q1,N,temp);
        mpz_cdiv_q((*A).Q1,(*A).Q1,(*A).Q0);

        while(!goodForm((*A),root))
                cFracStep(A,root);

        mpz_clear(temp);
}

//Given the last 2 steps in either the denominator or the numerater of the continued fraction convergent
//and b, calculates the next value and shifts R forward one step.
void approx(mpz_t R[2], mpz_t b)
{
    mpz_addmul(R[0],b,R[1]);
    mpz_swap(R[0],R[1]);
}

//Extended gcd algorithm.  Computes d = gcd(x,y,z) and ax+by+cz = d.
void Xgcd3(mpz_t d,mpz_t a, mpz_t b, mpz_t c, mpz_t x, mpz_t y, mpz_t z)
```

```
{
        mpz_t d_xy,a_xy;
        mpz_init(d_xy);
        mpz_init(a_xy);

        mpz_gcdext(d_xy,a,b,x,y);
        mpz_gcdext(d,a_xy,c,d_xy,z);

        mpz_mul(a,a,a_xy);
        mpz_mul(b,b,a_xy);

        mpz_clear(d_xy);
        mpz_clear(a_xy);
}

//Sets res = A*B, composition of quadratic forms.
void compose(struct form *res, struct form A, struct form B, mpz_t root,mpz_t N)
{
        mpz_t temp1,temp2,m,u,v,w;
        mpz_init(m);
        mpz_init(u);
        mpz_init(v);
        mpz_init(w);
        mpz_init(temp1);
        mpz_init(temp2);

        mpz_add(temp1,A.P,B.P);
        Xgcd3(m,u,v,w,A.Q0,B.Q0,temp1);

        mpz_mul(temp1,A.Q0,B.P);
        mpz_mul(temp1,temp1,u);
        mpz_mul(temp2,A.P,B.Q0);
        mpz_mul(temp2,temp2,v);
        mpz_add(temp1,temp1,temp2);
        mpz_set(temp2,N);
        mpz_addmul(temp2,A.P,B.P);
        mpz_mul(temp2,temp2,w);
        mpz_add(temp1,temp1,temp2);
        mpz_cdiv_q((*res).P,temp1,m);
        mpz_mul(temp1,A.Q0,B.Q0);
        mpz_mul(temp2,m,m);
        mpz_cdiv_q((*res).Q0,temp1,temp2);

        reduce(res,root,N);



        mpz_clear(m);
        mpz_clear(u);
        mpz_clear(v);
        mpz_clear(w);
        mpz_clear(temp1);
        mpz_clear(temp2);
}
```

B.2 Fast Return Functions

FastReturn was the name Shanks gave to the variation that used composition to test a perfect square quickly. This program provides the function that does this.

```
/* FastReturn.h
Used by SQUFOF to test whether a number is square and use saved quadratic forms to quickly find if a square
//form provides a factorization of N.
*/

#include "gmp.h"
#include<math.h>

int subIsSquare(mpz_t N, mpz_t root, int digit, mpz_t a)
{
    mpz_t testN,newa,B;
    mpz_init_set(testN,N);
    mpz_submul(testN,a,a);
    if(!mpz_sgn(testN))
    {
        mpz_set(root,a);
        mpz_clear(testN);
        return 1;
    }

    if(mpz_sgn(testN)<0)
    {
        mpz_clear(testN);
        return 0;
    }

    if(mpz_tstbit(testN,3*digit))
    {
        mpz_clear(testN);
        return 0;
    }

    int b = mpz_tstbit(testN,3*digit+1)+2*mpz_tstbit(testN,3*digit+2);
    int testa = mpz_tstbit(a,0)+2*mpz_tstbit(a,1)+4*mpz_tstbit(a,2);
    b = (b*testa)%4;

    mpz_clear(testN); //

    mpz_init_set_ui(B,b);
    mpz_init(newa);
    mpz_mul_2exp(B,B,3*digit);
    mpz_add(newa,a,B);
    int t = subIsSquare(N,root,digit+1,newa);
    if(t)
    {
        mpz_clear(B);
        mpz_clear(newa);
        return 1;
    }
    b = b+4;
    mpz_set_ui(B,b);
    mpz_mul_2exp(B,B,3*digit);
    mpz_add(newa,a,B);
    t = subIsSquare(N,root,digit+1,newa);
    mpz_clear(B);
    mpz_clear(newa);
    return t;
}
```

```
int isSquare(mpz_t N,mpz_t root)
{
    mpz_t lN;
    mpz_t a;
    int t;
    int twos = 0;
    double i;

    mpz_init_set(lN,N);
    while(!mpz_tstbit(lN,0))
    {
        twos++;
        mpz_fdiv_q_2exp(lN,lN,1); //Fix this
    }
    if(twos%2)
        return 0;
    twos = twos/2;
    mpz_init(a);

    if(mpz_tstbit(lN,1)||mpz_tstbit(lN,2))
        return 0;

    for(i = 1; i<8; i=i+2)
    {
        mpz_set_d(a,i);
        t = subIsSquare(lN,root,1,a);
        if(t)
        {
            mpz_clear(a);
            mpz_clear(lN);
            mpz_mul_2exp(root,root,twos);
            return 1;
        }
    }
    mpz_clear(a);
    mpz_clear(lN);
    return 0;
}
```

```
//Given that the first coefficient of A has sqrt = r and index i. saved[] is a list of previous forms
//such that saved[i] = F_{30*2^i}.  root = sqrt(N), where N is the discriminant (the number being factored).
//tr is set to 1 if a factor is found.
int testF(struct form A,struct form saved[],int i,mpz_t r,mpz_t root,mpz_t N,int *tr)
{
    struct form B,myA;
    mpz_t BP,AP,d;
    i = i/2;
    int size = 24;
    int os;

    int pow = 503316480; //30*2^24
    mpz_init_set(myA.Q0,A.Q0); mpz_init_set(myA.P,A.P); mpz_init_set(myA.Q1,A.Q1); mpz_init(myA.b);
    reverse(&myA,root);
    mpz_set(myA.Q0,r);
    reduce(&myA,root,N);
    while (size >= 0)
    {
        if(i>=pow)
        {
            i = i-pow;
            compose(&myA,myA,saved[size],root,N);
        }
        size = size-1;
        pow = pow/2;
    }

    mpz_init(B.Q0); mpz_init(B.P); mpz_init(B.Q1); mpz_init(B.b);
    ftof(&B,myA);
    reverse(&B,root);

    mpz_init(d);
    mpz_init_set_ui(BP,1);
    mpz_init_set_ui(AP,1);

    if(!mpz_cmp(myA.P,B.P))
    {
        mpz_gcd(d,myA.P,N);

        mpz_clear(BP); mpz_clear(AP);
        mpz_clear(B.Q0); mpz_clear(B.P); mpz_clear(B.Q1); mpz_clear(B.b);
        mpz_clear(myA.Q0); mpz_clear(myA.P); mpz_clear(myA.Q1); mpz_clear(myA.b);

        if(mpz_cmp_ui(d,2)>0)
        {
            mpz_out_str(stdout,10,d);
            mpz_clear(d);
            (*tr) = 1;
            return 1;
        }
        else
        {
            mpz_clear(d);
            (*tr) = 0;
            return 0;
        }
    }

    os = 0;

    while (mpz_cmp(myA.P,AP)&&mpz_cmp(B.P,BP))
    {
        mpz_set(AP,myA.P);
        mpz_set(BP,B.P);
        cFracStep(&myA,root);
        cFracStep(&B,root);
        os = os+1;
    }

    if (!mpz_cmp(myA.P,AP))
    {
        mpz_gcd(d,myA.Q0,N);

        mpz_clear(BP); mpz_clear(AP);
        mpz_clear(B.Q0); mpz_clear(B.P); mpz_clear(B.Q1); mpz_clear(B.b);
        mpz_clear(myA.Q0); mpz_clear(myA.P); mpz_clear(myA.Q1); mpz_clear(myA.b);

        if(mpz_cmp_ui(d,2)>0)
        {
            mpz_out_str(stdout,10,d);
            printf("\n");
            mpz_clear(d);
            (*tr)=1;
            return 1;
        }
        else
        {
            mpz_clear(d);
            (*tr)=0;
            return 0;
        }

    }

    if(!mpz_cmp(B.P,BP))
    {
        mpz_gcd(d,B.Q0,N);

        mpz_clear(BP); mpz_clear(AP);
        mpz_clear(B.Q0); mpz_clear(B.P); mpz_clear(B.Q1); mpz_clear(B.b);
        mpz_clear(myA.Q0); mpz_clear(myA.P); mpz_clear(myA.Q1); mpz_clear(myA.b);

        if(mpz_cmp_ui(d,2)>0)
        {
            mpz_out_str(stdout,10,d);
            printf("\n");
            mpz_clear(d);
            (*tr)=1;
            return 1;
        }
        else
        {
            mpz_clear(d);
            (*tr)=0;
            return 0;
        }
    }

    mpz_clear(BP); mpz_clear(AP);
    mpz_clear(B.Q0); mpz_clear(B.P); mpz_clear(B.Q1); mpz_clear(B.b);
```

```
    mpz_clear(myA.Q0); mpz_clear(myA.P); mpz_clear(myA.Q1); mpz_clear(myA.b);
    return 0;
}
```

B.3 Segments Parallelization

This code, parSQUFOFsegs.c, uses segments to distribute the cycle of quadratic forms between multiple processors. It is designed for up to 64 processors, although it would be easy to modify for a larger system or for distributed processing. It reads from standard output. Each processor also records some basic information about the factors it finds to a file named “1SXX.txt”, where “XX” is the two digit process number. At the end of the program, the master compiles all output into the file “SegmentsData.txt”. It has several other options.

-n=x specifies that there will be x integers to factor. -d specifies that some basic information should be output for each factor found. -D specifies that thorough information should be provided about the progress of each processor to assist in debugging. -i specifies that the computer should be interactive with the user

Thus, a standard format for the command would be:

mpirun -np 50 parSQUFOFsegs ¡numbers.txt -n=100 -d

which would use 50 processors to factor 100 integers from the file numbers.txt and output basic information about each to the screen.

```
//parSQUFOFsegs.c
//MIDN 1/C Steve McMath
//April, 2005

#include "mpi.h"
#include <stdio.h>
#include "NumberTheory.h"
#include "/usr/local/include/gmp.h"
#include <time.h>
#include "FastReturn.h"
```

```
#include "gmpmpi_segments.h"

int main(int argc, char *argv[])
{
  const int STOP = 10;
  const int finished = 11;
  const int segSize = 25;
  const int Npass1 = 12;
  const int Npass2 = 13;
  char fname[9] = "1S00.txt";
  int myrank,i,j,Ni,np,flag,done,tens,single;
  mpz_t N,root,firstBack,factor;
  struct form Start;
  struct form back[segSize];
  char *N_str;
  int lengthN;
  int dDi[3] = {0,0,0};
  int maxNi = 100;
  MPI_Status status;
  time_t start,endt;
  double dif;
  FILE *OUT,*lastOUT,*IN;

  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  MPI_Comm_size(MPI_COMM_WORLD, &np);

  //Idiot check
  if((np < 2) || (np > 50))
  {
    MPI_Finalize();
    return 0;
  }

  //Read in parameters from user.
  for(i = 1; i<argc; i++)
  {
    if(argv[i][0] == '-')
    {
```

```
    if(argv[i][1] == ’n’)
    {
      maxNi = 0;
      for(j = 3; argv[i][j]!=NULL; j++)
      {
        maxNi = 10*maxNi+(argv[i][j]-’0’);
      }
    }
    if(argv[i][1] == ’d’)
      dDi[0] = 1;
    if(argv[i][1] == ’D’)
    {
      dDi[0] = 1;
      dDi[1] = 1;
    }
    if(argv[i][1] == ’i’)
      dDi[2] = 1;
  }
}

//Set 2 digits in filename to processor number.
fname[2] = ’0’+myrank/10;
fname[3] = ’0’+myrank%10;

OUT = fopen(fname,"w");

for(Ni=0; Ni < maxNi; Ni++)
{

  fflush(OUT);
  flag = 0;
  done = 0;
  mpz_init(N);
  if(myrank==0) //Read in N and pass it to the other processors.
  {
    fprintf(OUT,"Starting %i\n",Ni);
    mpz_inp_str(N,stdin,10);
    lengthN = 1+mpz_sizeinbase(N,36);
    N_str = malloc(lengthN*sizeof(char));
    mpz_get_str(N_str,36,N);
  for(i = 1; i<np; i++)
  {
    MPI_Send(&lengthN,1,MPI_INT,i,Npass1,MPI_COMM_WORLD);
    MPI_Send(N_str,lengthN,MPI_CHAR,i,Npass2,MPI_COMM_WORLD);
  }
  free(N_str);
  time(&start);
}
else //Receive N from the master.
{
  MPI_Recv(&lengthN,1,MPI_INT,0,Npass1,MPI_COMM_WORLD,&status);
  N_str = malloc(lengthN*sizeof(char));
  MPI_Recv(N_str,lengthN,MPI_CHAR,0,Npass2,MPI_COMM_WORLD,&status);
  mpz_set_str(N,N_str,36);
  free(N_str);
  time(&start);
}

//Initialize memory and calculate first form.
mpz_init(root);
mpz_sqrt(root,N);

mpz_init_set_ui(Start.Q0,1);
mpz_init_set(Start.P,root);
mpz_init_set(Start.Q1,N);
mpz_submul(Start.Q1,root,root);
mpz_init(Start.b);

//Take 30 steps forward from the start.
mpz_init(firstBack);
for(i = 0; i<30; i++)
{
  cFracStep(&(Start),root);
  cFracStep(&(Start),root);
  if(isSquare(Start.Q0,firstBack)&&testF(Start,back,Start,0,firstBack,root,N,&done,factor,dDi))
  {
    mpz_clear(root);
    mpz_clear(Start.Q0);
    mpz_clear(Start.P);
    mpz_clear(Start.Q1);
    mpz_clear(Start.b);
    mpz_clear(N);
    MPI_Finalize();
    return 0;
  }
}

//Compute forms used by Fast Return.
for(i = 0; i<segSize; i++)
{
  mpz_init_set(back[i].Q0,Start.Q0);
  mpz_init_set(back[i].P,Start.P);
  mpz_init_set(back[i].Q1,Start.Q1);
  mpz_init(back[i].b);
  compose(&Start,Start,Start,root,N);
}

if (myrank ==0)
{
  int found[64];
  MPI_Request waiting[64],stop[64];
  struct segment Next;
  struct form hIncrement,nextHstart;

  for(i = 1; i<np; i++) //Receive messages from ready processors.
  {
    waiting[i] = 0;
    MPI_Irecv(&found[i],1,MPI_INT,i,finished,MPI_COMM_WORLD,&waiting[i]);
  }

  //hIncrement is used as the first step before Fast Return.
  mpz_init_set(hIncrement.Q0,back[segSize-1].Q0);
  mpz_init_set(hIncrement.P,back[segSize-1].P);
  mpz_init_set(hIncrement.Q1,back[segSize-1].Q1);
  mpz_init(hIncrement.b);

  //Next.start is the first form in the segment.
  mpz_init_set(Next.start.Q0,back[0].Q0);
  mpz_init_set(Next.start.P,back[0].P);
  mpz_init_set(Next.start.Q1,back[0].Q1);
```

```
mpz_init(Next.start.b);

//Next.end is the last form in the assigned segment.
mpz_init_set(Next.end.Q0,Start.Q0);
mpz_init_set(Next.end.P,Start.P);
mpz_init_set(Next.end.Q1,Start.Q1);
mpz_init(Next.end.b);

//halfStart is used in the first step before Fast Return.
mpz_init_set(Next.halfStart.Q0,back[0].Q0);
mpz_init_set(Next.halfStart.P,back[0].P);
mpz_init_set(Next.halfStart.Q1,back[0].Q1);
mpz_init(Next.halfStart.b);

mpz_init_set(nextHstart.Q0,hIncrement.Q0);
mpz_init_set(nextHstart.P,hIncrement.P);
mpz_init_set(nextHstart.Q1,hIncrement.Q1);
mpz_init(nextHstart.b);

//Records which segment this is.
Next.segnum = 0;
i = 1;
while(!done) //Checks for free processors, sends them a segment, then calculates the next.
{
  if(i>=np)
    i = 1;
  flag = 0;
  MPI_Test(&waiting[i],&flag,&status);
  if(flag)
  {
    if(found[i])
    {
      if(dDi[1])
        printf("Something found by rank %d\n",i);
      done = found[i];
    }
    else
    {
      Send(Next,i);
      if(dDi[1])
```

```
            printf("Segment %d given to rank %d\n",Next.segnum,i);
          Next.segnum++;
          MPI_Irecv(&found[i],1,MPI_INT,i,finished,MPI_COMM_WORLD,&waiting[i]);
          ftof(&Next.start,Next.end);
          ftof(&Next.halfStart,nextHstart);
          compose(&nextHstart,nextHstart,hIncrement,root,N);
          compose(&Next.end,nextHstart,nextHstart,root,N);
        }
      }
      i++;
    }

    //Let all the processors know when one finds a factor.
    for(i=1; i<np; i++)
      MPI_Isend(&done,1,MPI_INT,i,STOP,MPI_COMM_WORLD,&stop[i]);

    //Clear memory.
    mpz_clear(hIncrement.Q0); mpz_clear(hIncrement.P); mpz_clear(hIncrement.Q1); mpz_clear(hIncrement.b);
    mpz_clear(Next.start.Q0); mpz_clear(Next.start.P); mpz_clear(Next.start.Q1); mpz_clear(Next.start.b);
    mpz_clear(Next.end.Q0); mpz_clear(Next.end.P); mpz_clear(Next.end.Q1); mpz_clear(Next.end.b);
    mpz_clear(Next.halfStart.Q0); mpz_clear(Next.halfStart.P); mpz_clear(Next.halfStart.Q1);
     mpz_clear(Next.halfStart.b);
    mpz_clear(nextHstart.Q0); mpz_clear(nextHstart.P); mpz_clear(nextHstart.Q1); mpz_clear(nextHstart.b);

    //Wait for all processors to get the message.
    for(i=1; i<np; i++)
      MPI_Wait(&waiting[i],&status);
  }
  else
  {
    struct segment Mine;
    MPI_Request last;
    MPI_Request ready;
    long long int index;
    int test=0,found = 0,end = 0;

    //Initialize memory.
    mpz_init(Mine.start.Q0); mpz_init(Mine.start.P); mpz_init(Mine.start.Q1); mpz_init(Mine.start.b);
    mpz_init(Mine.end.Q0); mpz_init(Mine.end.P); mpz_init(Mine.end.Q1); mpz_init(Mine.end.b);
    mpz_init(Mine.halfStart.Q0); mpz_init(Mine.halfStart.P); mpz_init(Mine.halfStart.Q1);
```

```
 mpz_init(Mine.halfStart.b);

//Set up to receive the message when a processor finishes.
MPI_Irecv(&done,1,MPI_INT,0,STOP,MPI_COMM_WORLD,&last);
if(dDi[1])
  printf("%d, done = %d recieved\n",myrank,done);
while((!flag)&&(!end)) //Each cycle through this loop receives a segment from master.
{
  //Send 'ready' to master.
  MPI_Isend(&found,1,MPI_INT,0,finished,MPI_COMM_WORLD,&ready);

  //Receive a segment.
  Recv(&Mine);

  if((Mine.segnum==(np-2))&&dDi[1])
  {
    time(&endt);
    dif = difftime(endt,start);
    printf("All processes started in %e seconds.\n",dif);
  }
  reduce(&Mine.start,root,N);

  index = 0;
  //Basic step.  Checks for messages every cycle.
  //Does Fast Return if it finds a square.
  while(((!isSquare(Mine.start.Q0,firstBack))||(!testF(Mine.start,back,Mine.halfStart,index,firstBack,
   root,N,&test,factor,dDi)))&&(!end))
  {
    cFracStep(&Mine.start,root);
    cFracStep(&Mine.start,root);

    //Check for Start
    if((!mpz_cmp_ui(Mine.start.Q0,1))||(!mpz_cmp_ui(Mine.start.Q1,1)))
    {
      end = 1;
      found = -1;
      if(dDi[0])
        printf("Start found in segment %d",Mine.segnum);
    }
```

```
      //Check for end of segment.
      if((!mpz_cmp(Mine.start.Q0,Mine.end.Q0))&&(!mpz_cmp(Mine.start.P,Mine.end.P)))
        end = 1;

      //Check for message.
      MPI_Test(&last,&flag,&status);
      if(flag)
      {
        if(done==1)
          end = 1;
      }
      index++;
    }
    if(test)
    {
      found = 1;
      end = 1;
      time(&endt);
      dif = difftime(endt,start);
      if(dDi[0])
        printf("%i, Rank %i, %e, %i, ",Ni,myrank,dif,Mine.segnum);
      fprintf(OUT,"%i, Rank %i, %e, %i, ",Ni,myrank,dif,Mine.segnum);
      mpz_out_str(stdout,10,factor);
      printf("\n");
      fprintf(OUT,"\n");
    }
    MPI_Test(&last,&flag,&status);
  }
  MPI_Isend(&found,1,MPI_INT,0,finished,MPI_COMM_WORLD,&ready);

  mpz_clear(Mine.start.Q0); mpz_clear(Mine.start.P); mpz_clear(Mine.start.Q1); mpz_clear(Mine.start.b);
  mpz_clear(Mine.end.Q0); mpz_clear(Mine.end.P); mpz_clear(Mine.end.Q1); mpz_clear(Mine.end.b);
  mpz_clear(Mine.halfStart.Q0); mpz_clear(Mine.halfStart.P); mpz_clear(Mine.halfStart.Q1);
   mpz_clear(Mine.halfStart.b);
}

time(&endt);
dif = difftime(endt,start);
if(dDi[1])
  printf("Rank = %d: Done in %e sec\n",myrank,dif);
  //Clear memory used.
  for(i=0; i<segSize; i++)
  {
    mpz_clear(back[i].Q0);
    mpz_clear(back[i].P);
    mpz_clear(back[i].Q1);
    mpz_clear(back[i].b);
  }

  mpz_clear(Start.Q0);
  mpz_clear(Start.P);
  mpz_clear(Start.Q1);
  mpz_clear(Start.b);
  mpz_clear(N);
  mpz_clear(root);
}

fclose(OUT);

if(myrank==0) //Compile the information from the different processors into a single file.
{
  lastOUT = fopen("SegmentsData.txt","w");

  for(i=0; i<np; i++)
  {
    tens = i/10;
    fname[2] = '0'+tens;
    fname[3] = '0'+i%10;
    IN = fopen(fname,"r");
    single = fgetc(IN);
    do
    {
      fprintf(lastOUT,"%c",single);
      single = fgetc(IN);
    } while(single!=EOF);
    fprintf(lastOUT,"\n");
    fclose(IN);
    fflush(lastOUT);
    }
    fclose(lastOUT);
  }

  MPI_Finalize();
  return 0;
}
```

B.3.1 Segments Communication Functions

This file, gmpmpi_segments.h, is used by parSQUFOFsegs.c to send the data related to a segment from the master processor to the other processors.

```
struct segment {
    struct form start;
    struct form end;
    struct form halfStart;
    int segnum;
};

void Send(struct segment A, int dest);
void Recv(struct segment *A);

//Determines the lengths of each intiger in base 36 and allocates space for them to be stored as integers.  The
//lengths can be sent to the other processor to allocate space and be able to receive.
void form_type(struct segment A, char *A_str[6], int lengths[6])
{
    int i;

    lengths[0] = 1+mpz_sizeinbase(A.start.Q0,36);
    lengths[1] = 1+mpz_sizeinbase(A.start.P,36);
    lengths[2] = 1+mpz_sizeinbase(A.end.Q0,36);
    lengths[3] = 1+mpz_sizeinbase(A.end.P,36);
    lengths[4] = 1+mpz_sizeinbase(A.halfStart.Q0,36);
    lengths[5] = 1+mpz_sizeinbase(A.halfStart.P,36);

    for(i = 0; i<6; i= i+1)
    {
      A_str[i] = malloc(lengths[i]*sizeof(char));
    }

    mpz_get_str(A_str[0],36,A.start.Q0);
    mpz_get_str(A_str[1],36,A.start.P);
    mpz_get_str(A_str[2],36,A.end.Q0);
    mpz_get_str(A_str[3],36,A.end.P);
    mpz_get_str(A_str[4],36,A.halfStart.Q0);
    mpz_get_str(A_str[5],36,A.halfStart.P);
}

//allocates space for the lengths sent
void reform_type(char *A_str[6],int lengths[6])
{
    int i;
    for(i = 0; i<6; i++)
      A_str[i] = malloc(lengths[i]*sizeof(char));
}

//After receiving the strings, converts them to mpz and stores them correctly.
void unpack(struct segment *A, char *A_str[6])
{
    mpz_set_str((*A).start.Q0,A_str[0],36);
    mpz_set_str((*A).start.P,A_str[1],36);
    mpz_set_str((*A).end.Q0,A_str[2],36);
    mpz_set_str((*A).end.P,A_str[3],36);
    mpz_set_str((*A).halfStart.Q0,A_str[4],36);
    mpz_set_str((*A).halfStart.P,A_str[5],36);
}

//Sends a segment A to processor #dest
void Send(struct segment A, int dest)
{
  char *As[6];
  int lengths[6];
  const int tags[6] = {1,2,3,4,5,6};
  const int extra = 7;
  int i;



  form_type(A,As,lengths);
  MPI_Send(&lengths[0],6,MPI_INT,dest,0,MPI_COMM_WORLD);
  for(i = 0; i<6; i++)
    MPI_Send(As[i],lengths[i],MPI_CHAR,dest,tags[i],MPI_COMM_WORLD);

  MPI_Send(&(A.segnum),1,MPI_INT,dest,extra,MPI_COMM_WORLD);
  for(i = 0; i<6; i++)
    free(As[i]);
}

//Receive segment A from processor 0
void Recv(struct segment *A)
{
  char *As[6];
  int lengths[6];
  const int tags[6] = {1,2,3,4,5,6};
  int i;
  MPI_Status halt;
  const int extra = 7;

  MPI_Recv(&lengths[0],6,MPI_INT,0,0,MPI_COMM_WORLD,&halt);
  reform_type(As,lengths);
  for(i = 0; i<6; i++)
    MPI_Recv(As[i],lengths[i],MPI_CHAR,0,tags[i],MPI_COMM_WORLD,&halt);
  unpack(A,As);
  MPI_Recv(&((*A).segnum),1,MPI_INT,0,extra,MPI_COMM_WORLD,&halt);
  for(i=0; i<6; i++)
    free(As[i]);
}
```

B.4 Multipliers Parallelization

This code, parSQUFOFmults.c, uses different multipliers to distribute the factorization algorithm between multiple processors. It is currently configured for up to 50 processors, although it would be easy to modify it for a larger system or for distributed processing. Commands, input, and output are the same as parSQUFOFsegs, except that the files are named “2SXX.txt” and the final file output is named “MultipliersData.txt”.

```
//parSQUFOFmults.c
//MIDN 1/C Stephen McMath
//May 2005

#include "mpi.h"
#include <stdio.h>
#include "NumberTheory.h"
#include "/usr/local/include/gmp.h"
#include <time.h>
#include "FastReturn.h"

#define NUMPROC 51 //Number of processors configured for.

//Single implementation of SQUFOF.
int SQUFOF(mpz_t N,mpz_t factor,int dDi[3]);

int main(int argc, char *argv[])
{
  //Constants for sending message to stop working.
  const int STOP[64] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,
   32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63};
  const int Npass1 = 12;  //Constants for passing N from master to slaves.
  const int Npass2 = 13;
  int myrank,np,i,j,success;
  int dDi[3] = {0,0,0}; //d for basic information.  D for thorough debugging.  i for interactive
  int maxNi = 100; //Can also be changed by user.
  mpz_t N, factor, temp;
  char *N_str; //Used for passing N from the master.
  int lengthN;
  char fname[9] = "2S00.txt"; //Stores basic information from each processor.
  MPI_Request stop[NUMPROC];
  //Multiples chosen to minimize size of largest prime used and arranged in increasing order.
  int mults[NUMPROC] = {1,2,3,5,6,7,10,11,13,14,15,21,22,26,30,33,35,39,42,55,65,66,70,77,78,91,105,110,130,
   143,154,165,182,195,210,231,273,286,330,338,385,390,429,455,462,546,715,770,858,1155,2310};
  MPI_Status status;
  time_t start,endt;
  double dif;
  int Ni,tens;
  int single;
  FILE *OUT;
  FILE *IN, *lastOUT;

  time(&start);
  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  MPI_Comm_size(MPI_COMM_WORLD, &np);

  //Idiot check
  if((np < 2) || (np > NUMPROC))
  {
    MPI_Finalize();
    return 0;
  }

  //Read in parameters from user.
  for(i = 1; i<argc; i++)
  {
    if(argv[i][0] == '-')
    {
      if(argv[i][1] == 'n')
      {
        maxNi = 0;
        for(j = 3; argv[i][j]!=NULL; j++)
        {
          maxNi = 10*maxNi+(argv[i][j]-'0');
        }
      }
      if(argv[i][1] == 'd')
        dDi[0] = 1;
      if(argv[i][1] == 'D')
      {
        dDi[0] = 1;
        dDi[1] = 1;
      }
      if(argv[i][1] == 'i')
        dDi[2] = 1;
    }
```

```
}

mpz_init(N);

//Set 2 digits in filename to processor number.
tens = myrank/10;
fname[2] = ’0’+tens;
fname[3] = ’0’+myrank%10;

OUT = fopen(fname,"w");

for(Ni=0; Ni<maxNi; Ni++)
{

  fflush(OUT);

  if(myrank==0)  //Read in N and pass it to the other processors.
  {
      if(dDi[2]==1)
        printf("N: \n");
      mpz_inp_str(N,stdin,10);
      lengthN = 1+mpz_sizeinbase(N,36);
      N_str = malloc(lengthN*sizeof(char));
      mpz_get_str(N_str,36,N);
      for(i = 1; i<np; i++)
      {
        MPI_Send(&lengthN,1,MPI_INT,i,Npass1,MPI_COMM_WORLD);
        MPI_Send(N_str,lengthN,MPI_CHAR,i,Npass2,MPI_COMM_WORLD);
      }
      free(N_str);
      fprintf(OUT,"Starting Ni = %d\n",Ni);
      time(&start);
  }
  else  //Receive N from the master.
  {
    MPI_Recv(&lengthN,1,MPI_INT,0,Npass1,MPI_COMM_WORLD,&status);
    N_str = malloc(lengthN*sizeof(char));
    MPI_Recv(N_str,lengthN,MPI_CHAR,0,Npass2,MPI_COMM_WORLD,&status);
    mpz_set_str(N,N_str,36);
    free(N_str);
    time(&start);
  }

  //Each processor multiplies N by its multiple.
  mpz_mul_ui(N,N,mults[myrank]);

  if(dDi[1])
    printf("Rank %d, *%d\n",myrank,mults[myrank]);

  mpz_init(factor);

  //Apply standard SQUFOF algorithm (with Fast Return) to the new N.
  success = SQUFOF(N,factor,dDi);

  if(success)  //Output information according to user parameters.
  {
    mpz_init(temp);
    mpz_gcd_ui(temp,factor,mults[myrank]);
    mpz_fdiv_q(factor,factor,temp);
    mpz_clear(temp);
    if(dDi[1])
      printf("Rank %d, success!\n",myrank);
    time(&endt);
    dif = difftime(endt,start);
    fprintf(OUT,"%d, %d, %e, ",Ni,mults[myrank],dif);
    if(dDi[0])
    {
      printf("Ni: %d, mult: %d, time: %e sec, factor: ",Ni,mults[myrank],dif);
      mpz_out_str(stdout,10,factor);
      printf("\n");
    }

    mpz_out_str(OUT,10,factor);
    fprintf(OUT,"\n");
    for(i = 0; i<np; i++) //Inform the other processors.
    {
      if(i!=myrank)
        MPI_Isend(&success,1,MPI_INT,i,STOP[myrank],MPI_COMM_WORLD,&stop[i]);
    }
    for(i=0; i<np; i++) //Wait for each processor to receive the message.
```

```
    {
      if(i!=myrank)
        MPI_Wait(&stop[i],&status);
    }
  }

  time(&endt);
  if(dDi[1])
  {
    dif = difftime(endt,start);
    printf("Rank %d, done in %e\n",myrank,dif);
  }

  if(myrank!=0) //Inform master when ready for next N.
    MPI_Isend(&success,1,MPI_INT,0,-1,MPI_COMM_WORLD,&stop[0]);
  else
  {
    for(i=1; i<np; i++)
      MPI_Recv(&success,1,MPI_INT,i,-1,MPI_COMM_WORLD,&status);
  }


}

mpz_clear(N);
mpz_clear(factor);

fclose(OUT);

if(myrank==0) //Compile the information from the different processors into a single file.
{
  lastOUT = fopen("MultipliersData.txt","w");

  for(i=0; i<np; i++)
  {
    tens = i/10;
    fname[2] = '0'+tens;
    fname[3] = '0'+i%10;
    IN = fopen(fname,"r");
    single = fgetc(IN);
```

```
      do
      {
        fprintf(lastOUT,"%c",single);
        single = fgetc(IN);
      } while(single!=EOF);
      fprintf(lastOUT,"\n");
      fclose(IN);
      fflush(lastOUT);
    }
    fclose(lastOUT);
  }
  MPI_Finalize();
  return 0;
}



//Only difference between this and the standard algorithm (with Fast Return) is the check for a message
//from another processor indicating completion.
int SQUFOF(mpz_t N, mpz_t factor, int dDi[3])
{
    const int STOP[64] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,
       32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63};
    mpz_t root;
    struct form A,firstA;
    struct form saved[30];
    mpz_t r;
    int found[NUMPROC];
    MPI_Request waiting[NUMPROC];
    long long i;
    int pow=30,size=0,success=0, stop=1000000000;
    int np,myrank,flag=0,j,k;
    MPI_Status status;

    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    for(i = 0; i<np; i++) //Prepare to receive messages from the other processes.
    {
      if(i!=myrank)
        MPI_Irecv(&found[i],1,MPI_INT,i,STOP[i],MPI_COMM_WORLD,&waiting[i]);
```

```
}

//Initialize memory and calculate first form.
mpz_init(root);
mpz_sqrt(root,N);

mpz_init_set_ui(A.Q0,1);
mpz_init_set(A.P,root);
mpz_init(A.Q1);
mpz_init(A.b);
mpz_init_set_ui(r,1);

reduce(&A,root,N);

mpz_init(firstA.Q0); mpz_init(firstA.P); mpz_init(firstA.Q1); mpz_init(firstA.b);
ftof(&firstA,A);

cFracStep(&A,root);
cFracStep(&A,root);
i = 0;

//This is the basic step in the algorithm.
//Searches the given segment for a square.  Uses Fast Return to test any squares found.
//Checks each cycle for a message from another processor.
while(((!isSquare(A.Q0,r))||(!testF(A,saved,firstA,i,r,root,N,&success,factor,dDi)))&&(i<stop)&&(!flag))
{
  if(i==pow)
  {
    mpz_init(saved[size].Q0); mpz_init(saved[size].P);
    mpz_init(saved[size].Q1); mpz_init(saved[size].b);
    ftof(&saved[size],A);
    size = size+1;
    pow = 2*pow;
  }
  cFracStep(&A,root);
  cFracStep(&A,root);
  if((!mpz_cmp_ui(A.Q0,1))||(!mpz_cmp_ui(A.Q1,1)))
    i = stop;
  i = i+1;
  for(j = 0; (j<np)&&(!flag); j++)
  {
    if(j!=myrank)
      MPI_Test(&waiting[j],&flag,&status);
    if(flag)
    {
      if(dDi[1])
        printf("Rank %d flag by %d\n",myrank,j);
      for(k=0; k<np; k++)
      {
        if((k!=j)&&(k!=myrank))
          MPI_Cancel(&waiting[k]);
      }
    }
  }
}

//Cancel unused communications.
for(j = 0; (j<np)&&(!flag); j++)
{
  if(j!=myrank)
    MPI_Test(&waiting[j],&flag,&status);
  if(flag)
  {
    for(k=0; k<np; k++)
    {
      if((k!=j)&&(k!=myrank))
        MPI_Cancel(&waiting[k]);
    }
    success = 0;
  }
}

//Tell the other processors about the factor found.
if(success)
{
  for(k = 0; k<np; k++)
  {
    if(k!=myrank)
      MPI_Cancel(&waiting[k]);
  }
```

```
      if(dDi[1])
        printf("Found by rank %d\n",myrank);
    }

    //Clear memory used.
    for(i = 0; i<size; i++)
    {
      mpz_clear(saved[i].Q0);
      mpz_clear(saved[i].P);
      mpz_clear(saved[i].Q1);
      mpz_clear(saved[i].b);
    }

    mpz_clear(A.Q0);
    mpz_clear(A.P);
    mpz_clear(A.Q1);
    mpz_clear(A.b);

    mpz_clear(firstA.Q0);
    mpz_clear(firstA.P);
    mpz_clear(firstA.Q1);
    mpz_clear(firstA.b);

    mpz_clear(r);
    return success;
}
```